Package ‘comsimitv’

October 12, 2022

Type Package
Title Flexible Framework for Simulating Community Assembly
Version 0.1.5
Author Zoltan Botta-Dukat
Maintainer Zoltan Botta-Dukat <botta-dukat.zoltan@okologia.mta.hu>
Description Flexible framework for trait-based simulation of community assembly, where components could be replaced by user-defined function and that allows variation of traits within species.
Imports MASS, vegan
License GPL-2
Encoding UTF-8
RoxygenNote 7.1.1
Suggests knitr, markdown, rmarkdown, testthat, bookdown
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2021-07-17 16:30:02 UTC

R topics documented:

asymmetric.competition.kernel ........................................... 2
comm.sampling ............................................................. 3
comm.simul ................................................................. 4
competition.kernel ......................................................... 6
fDispersal ................................................................. 7
fITV ....................................................................... 8
Gaussian.competition.kernel ............................................. 8
Gaussian.tolerance ......................................................... 9
Gener.species.pool ......................................................... 10
MetaCom.Dispersal ......................................................... 11
asymmetric.competition.kernel

Description

It calculates asymmetric competition coefficients.

Usage

asymmetric.competition.kernel(
  trait.values,
  trait.compet = "trait.b",
  ac.type = c("Kisdi", "Nattrass"),
  sigma.b = 0.03,
  ac.C = 1,
  ac.v = 1,
  ...
)

Arguments

 trait.values Dataframe of all traits
 trait.compet Name of trait related to resource use
 ac.type Type of the function (see vignette("competition"))
 sigma.b steepness of competition kernel
 ac.C parameter influencing shape of the function (has to be positive)
 ac.v parameter influencing shape of the function (has to be positive)
 ... Any additional parameters

Details

Depending on value of ac.type the convex-concave function from Kisdi (1999) or smooth function suggested by Nattrass et al (2012) are used.

For formulas and meaning of parameters see the vignette("competition")
References


See Also

*competition.kernel*

---

**comm.sampling**

*Converting simulation results into site-by-species matrix*

**Description**

Converts simulation result into site-by-species matrix of abundances, and optionally in the same step simulates random sampling with fixed number of individuals.

**Usage**

```
comm.sampling(x, type = c("full", "random"), size)
```

**Arguments**

- **x**: community and trait data matrix produced by `comm.simul` function
- **type**: Type of sampling. If `type="full"` sample size equals to community size. It simply converts `x` into a site-by-species matrix. If `type="random"`, it applies random sampling by calling (`rrarefy`) function
- **size**: Number of individuals in the random samples. It should be smaller than number of individuals in simulated (sub)communities. Otherwise, `x` is converted into a site-by-species matrix without (re)sampling.

**Details**

If `type="full"` it simply converts simulation results from long to wide format. If `type="random"` it randomly selects `size` individuals in each (sub)community and abundances in these samples are converted into site-by-species matrix format.

**Value**

A site-by-species matrix containing abundances.
Examples

```r
x <- comm.simul(S = 20, J = 30)
str(x$final.community)

w <- comm.sampling(x$final.community, type = "full")
str(w)

w.rarefied <- comm.sampling(x$final.community, type = "random", size = 10)
rowSums(w)
rowSums(w.rarefied)
```

---

**comm.simul**  
**Framework for community assembly simulation**

**Description**

Flexible framework of individual-based simulation of community assembly following framework proposed by Botta-Dukat & Czucz (2016), but allowing intraspecific trait variation (ITV)

**Usage**

```r
comm.simul(
  x = vector(),
  S = 200,
  n.traits = 3,
  J = 300,
  rand.seed = NULL,
  sim.length = 1,
  fSpecPool = "Gener.species.pool",
  competition.kernel = "Gaussian.competition.kernel",
  fSurvive = "Gaussian.tolerance",
  fSeedProduction = "SeedProduction",
  fDispersal = "MetaCom.Dispersal",
  fITV = "randomITV",
  verbose = FALSE,
  ...
)
```

**Arguments**

- **x**  
  Vector of environmental values in communities. If not given, 40 communities are created, with environmental variable equally spacing from 0.11 to 0.89
- **S**  
  Species pool size
- **n.traits**  
  Number of traits
- **J**  
  Number of individuals in each community
**Details**

This function is a framework for simulation of assembly in a meta-community. The simulation consists of a community initialization followed by an iterative simulation of a "disturbance–regeneration" cycle. During initialization a species pool is created defining each species by its trait values. Each locality is characterized by an environmental variable. Initial composition of local communities is a random selection from the species pool: species identity is selected independently for each individual with probability of seedling survival (that depends on local environment and trait value).

The "disturbance-regeneration" cycle consists of the following steps:

1. disturbance event: some randomly selected individuals die in each community
2. survivors produce seeds. Seed production depends on fertility of the locality and competition among coexisting individuals
3. seeds are dispersed among localities
4. all seeds germinate and seedlings struggle for survival. The number of adults in local communities is fixed, thus number of seedlings that can survive and grow up equals to the number of individuals died in the disturbance event (in the recent version one individual dies, but planned development is introducing a disturbance severity/number of deaths parameter)

It is a flexible framework that calls functions for:

- **rand.seed** Random seed number. Setting the same value allows repeating the same simulation
- **sim.length** Length of simulation. \(\text{sim.length} \times S\) cycle (disturbance-seed production-dispersal-establishment) will happen.
- **fSpecPool** Name of (the user defined) function that generates the species pool. See `Gener.species.pool`
- **competition.kernel** Name of the (user defined) function for calculating pairwise competition coefficients. See more details in available functions and specification of your own function in `competition.kernel`
- **fSurvive** Name of the (user defined) function for calculating survival probability of seeds. See more details in available functions and specification of your own function in `tolerance`
- **fSeedProduction** Name of the user defined function for calculating number of produced seeds See `SeedProduction`
- **fDispersal** Name of the user defined function for dispersal of produced seeds among local communities. See more details in available functions and specification of your own function in `fDispersal`
- **fITV** Name of the function that define seeds trait values, possibly considering mother’s trait and mothers environment. If "noITV", there is no intraspecific trait variation. See more details in available functions and specification of your own function in `fITV`
- **verbose** Running may take long time. If verbose set to TRUE, it writes messages into the screen indicating the progress.
- **...** Additional parameters of functions called by the framework.
• generating species pool (Gener.species.pool)
• calculating pairwise competition coefficients (competition.kernel)
• calculating seedling’s survival probabilities (tolerance)
• calculating number of produced seeds (SeedProduction)
• calculating trait values of offsprings (fITV)
• seed dispersal among localities (fDispersal)

Functions available in the package can be easily replaced by user-defined functions.

Value

A list with two elements:
$final.community a dataframe containing data on individuals in the final meta-community. Each individual represented by a row; columns are: sub-community, species identity, trait values.
$parameters list of simulation parameters (including parameters of functions called by the framework function)

References


Examples

w<-comm.simul(S=20, J=30)
str(w)
set.seed(1)
w<-comm.simul(S=20, J=30, fITV=NULL)$final.community
w[w[,2]==1,] # Each individuals belonging to Species1 has the same trait values

competition.kernel Competition kernels

Description

User defined functions for calculating pairwise competition coefficients

Arguments

trait.values Values of trait related to resource use
... Additional parameters
Details

User can define any specific form of competition. Pairwise competition between species/individuals should depend on their trait values related to resource use. Vector of these trait values has to be the first parameter of the function, and any further parameters are allowed. The output has to be a square matrix of pairwise competition coefficients.

Competition kernels available in the package:
- asymmetric.competition.kernel
- Gaussian.competition.kernel

Value

Square matrix of pairwise competition coefficients

---

**fDispersal**

*User defined functions for dispersal*

Description

These functions define how seeds can spread among local communities.

Arguments

before          matrix where each seed is represented by one row, and seeds’s attributes (location, species, trait values) are in the columns
...             Additional parameters of functions called by the framework.

Details

User can define any rule for seed dispersal. The only requirement is that both first argument and value of the function should be a matrix where each seed is represented by one row, and seeds’s attributes (location, species, trait values) are in the columns. The locality information has to be stored in column named ‘site’.

Available function in the package:
- MetaCom.Dispersal

Value

Same type as the first argument.
Intraspecific Trait Variation

Description
User defined function for Intraspecific Trait Variation

Arguments
seeds Matrix of produced seeds (with mother's trait values) as produced by SeedProduction function
...

Other parameters of the function

Details
User can defined any specific function for ITV, e.g. random variation around mothers value, or maternal effect.
The first parameter has to be matrix of produced seeds, in the form as it created by SeedProduction function, and the results has to be in the same matrix form with updated trait values.
ITV functions available in the package:
randomITV

Value
The same type as seeds parameter, i.e. a matrix where each seed is represented by one row, and seeds's attributes (location, species, trait values) are in the columns

Gaussian competition kernel

Description
It calculates pairwise competition coefficients as overlap of Gaussian resource utilization curve

Usage
Gaussian.competition.kernel(
    trait.values,
    trait.compet = "trait.b",
    sigma.b = 0.03,
    ...
)
Arguments

- **trait.values**: Dataframe of all traits
- **trait.compet**: Name of trait related to resource use
- **sigma.b**: Width of Gaussian kernel
- **...**: Any additional parameters

Details

It assumes that each species has Gaussian resource utilization curve:

\[ \exp\left( \frac{(x - \text{trait.value})^2}{\text{sigma.b}} \right) \]

where: \( x \) = quality of resource (e.g. seed size or rooting depth)

Optima of curves depend on trait value related to resource use, while standard deviation is the same for all species (note that for technical reason parameter \( \text{sigma.b} \) is twice of the common squared s.d.). Pairwise competition coefficients are calculated as overlap of resource utilization functions (MacArthur & Levins 1967). See details in vignette("competition")

References


See Also

- competition.kernel

Description

It calculates probability of seedling’s survival from their trait related to habitat filtering and the local environment.

Usage

```r
Gaussian.tolerance(
  trait.values, 
  env, 
  env.trait = "trait.a", 
  sigma.a = 0.001, 
  ... 
)
```
Gener.species.pool

Arguments

- **trait.values**: Dataframe of all traits
- **env**: Vector of environmental conditions in the local communities
- **env.trait**: Name of trait related to environmental tolerance
- **sigma.a**: Tolerance width (same for all species)
- **...**: Any additional parameters

Details

It assumes that probability of seedling's survival is maximal if the local environment has the same value as its trait. Survival probability decrease as environmental value departs from the optimum according to a Gaussian (bell-shaped) curve. The speed of decrease depends on the tolerance width parameter (sigma.a).

Value

A matrix of survival probabilities, communities in rows, species/individuals in columns

See Also

tolerance

Gener.species.pool

Generating trait values for the species pool

Description

It generates random trait values for species. Each species (individual) are characterized by three traits.

Usage

Gener.species.pool(
  S,
  n.traits = 3,
  distribs = rep("unif", n.traits),
  distr.parms = list(),
  sigma = diag(1, n.traits, n.traits),
  ...
)

**Arguments**

- `S`  
  Species pool size

- `n.traits`  
  Number of traits

- `distribs`  
  Types of the distributions of traits

- `distr.parms`  
  Parameters of distribution (see Details)

- `sigma`  
  Matrix of variance-covariance matrix of traits

- `...`  
  Any additional parameters

**Details**

Each species are characterized by three traits called trait A, B and C. Trait A describes the habitat preference, trait B influences the competitive interactions, while trait C is a completely neutral trait.

Any standard distribution of stats package can be used for generating the random numbers. For list of these distribution see [Distributions](#). In stats package the functions for the density/mass function are named in the form `dxxx.*xxx*` (without d!) as string (i.e. between quotation marks) should be supplied for parameter `distribs`.

In this step single value of each trait is generated for each species, i.e. there is no intraspecific trait variation.

If traits are independent (it is the default option), random number generating functions are called with parameters specified by the user.

Otherwise, a variance-covariance matrix has to be given. First, triplets of random numbers are drawn from multivariate normal distribution with zero means and the supplied variance-covariance matrix as parameters. Then these random numbers are converted to probability by standard normal probability function, and then these probabilities converted to trait values using quantile function of selected distribution with parameters given by the user.

**Value**

A data frame with traits as columns

---

**MetaCom.Dispersal** | Seed dispersion in a metacommunity

**Description**

Seeds can disperse to any other local community with the same probability; i.e. probability to disperse other subcommunity/(number local communities - 1). Each seed is dispersed independently.

**Usage**

```r
MetaCom.Dispersal(n, before, m = 0.1, ...)
```
Arguments

n  number of local (sub)communities

before  A matrix of seed’s attributes; seeds in rows, their location, species identity and traits are in columns. Column that contains information on locality has to be called 'site'

m  probability that a seed are dispersed into other (sub)community

...  Additional parameters. It necessary for thechnical reasons: the framework don’t know the current list of parameters when call this function

Details

Both input and output is a matrix where seeds are in the rows, and their attributes (i.e. location, species identity and trait values) are in the columns.

Value

Same type of matrix as before

See Also

fDispersal

Description

This function adds a random noise to mother’s trait values of each seed

Usage

randomITV(
  seeds = matrix(),
  n.traits = 3,
  distribs = rep("unif", n.traits),
  distr.parms = list(),
  sigma = diag(1, n.traits, n.traits),
  ITV.ratio = 0.01,
  ...
)
SeedProduction

Arguments

- **seeds**: Matrix of produced seeds (with mother’s trait values) as produced by `SeedProduction` function
- **n.traits**: Number of traits
- **distrib**: Types of the distributions of traits (see `Gener.species.pool`)
- **distr.parms**: Parameters of distribution (see `Gener.species.pool`)
- **sigma**: Matrix of variance-covariance matrix of traits (see `Gener.species.pool`)
- **ITV.ratio**: Ratio of within/between species variances of traits
- **...**: Any additional parameters

Details

The function uses parameters of `Gener.species.pool`. First it transforms back mother’s trait values to multivariate normal distribution. Then random noise was added to this values. Random noise has multivariate normal distribution, with zero means and the same correlation structure as specified in parameter `sigma`. Note that `sigma` specifies covariance matrix, not correlation structure per se. Variances in the random noise are diagonals (i.e. variance components) of parameter `sigma` multiplied by `ITV.ratio`. The non-diagonal elements of covariance matrix were specified to conserve the correlation structure among traits.

Value

Matrix of produced seeds as produced by `SeedProduction` function

---

SeedProduction  Calculating number of produced seeds

Description

Number of seeds calculated following the formula used by Botta-Dukat & Czucz (2016). This built-in function can be replaced by a user-defined one.

Usage

```
SeedProduction(compet, b0 = 1, K = 200, seed.distrib = c("pois", "binom"), ...)
```

Arguments

- **compet**: Matrix of pairwise competition coefficients
- **b0**: Probability of producing seed, if no competition
- **K**: Critical level of competition (See Details)
- **seed.distrib**: Distribution of seed numbers (See Details)
- **...**: any additional parameters
Details

Expected value of produced seeds is a decreasing sigmoid function of strength of competition (sum of abundances weighted by competition coefficients). If strength of competition is higher than parameter K, probability is set to zero. See vignette("competition") for formulas.

In simulation of Botta-Dukat & Czucz (2016) each individual produces one seed or does not produce seed at all. In this case number of seeds follows binomial distribution (i.e. distrib="binom"). A more realistic alternative is using Poisson distribution (distrib="pois").

Value

Matrix of produced seeds

<table>
<thead>
<tr>
<th>tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Habitat suitability (tolerance) functions</td>
</tr>
</tbody>
</table>

Description

User defined functions for habitat suitability.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trait.values</td>
<td>Values of trait related to habitat filtering</td>
</tr>
<tr>
<td>env</td>
<td>Vector of environmental conditions in the local communities</td>
</tr>
<tr>
<td>...</td>
<td>Additional parameters</td>
</tr>
</tbody>
</table>

Details

User can define any specific function of habitat suitability, depending on environmental conditions and trait value related to habitat filtering. Vectors of these trait values and environmental conditions have to be the first and second parameter of the function, and any further parameters are allowed. The output has to be a matrix of habitat suitabilities, communities in rows, species/individuals in columns.

Tolerance functions available in the package:

Gaussian.tolerance

Value

A matrix of habitat suitabilities, communities in rows, individuals in columns.
trait.sampling

Simulated sampling for trait value measurement

Description
Randomly selects individuals for trait value measurement and gives back raw measured traits or their means.

Usage
trait.sampling(x, ITV = FALSE, aggregate = TRUE, n = 5)

Arguments
- x: community and trait data matrix produced by `comm.simul` function
- ITV: If TRUE each subcommunity are sampled separately, otherwise the meta-community level sampling was done.
- aggregate: If TRUE mean trait values are returned, otherwise the raw values of sampled individuals.
- n: Number of sampled individuals.

Details
It simulates the real world situation that not all individuals are collected for trait measurement. If ITV==FALSE, all individuals belonging to the species are pooled, and then n randomly selected individuals are measured. If ITV==TRUE, n individuals are measured in each (sub)community, where the species occur. If the occurring individuals are less than n, all individuals are measured. If aggregate==TRUE, meta-community or subcommunity level means are calculated, otherwise raw measurements are returned.

Value
data.frame with fields: species, site (only if ITV=TRUE), trait.a, trait.b, trait.c (raw values or means depending on parameter aggregate)

Examples
```r
x<-comm.simul(S=20, J=30)
str(x)

w<-trait.sampling(x$final.community)
w

w<-trait.sampling(x$final.community,ITV=TRUE,aggregate=TRUE)
str(w)
```
Index

asymmetric.competition.kernel, 2, 7

comm.sampling, 3
comm.simul, 3, 4, 15
competition.kernel, 3, 5, 6, 6, 9

Distributions, 11

fDispersal, 5, 6, 7, 12
fITV, 5, 6, 8

Gaussian.competition.kernel, 7, 8
Gaussian.tolerance, 9, 14
Gener.species.pool, 5, 6, 10, 13

MetaCom.Dispersal, 7, 11

randomITV, 8, 12
rrarefy, 3

SeedProduction, 5, 6, 8, 13, 13

tolerance, 5, 6, 10, 14
trait.sampling, 15