Package: fdcoexist (via r-universe)

August 31, 2024

Title Multi-Species Trait-Based Coexistence Model in Discrete time

Version 0.0.1

Description A modified Beverton-Holt model used in the Denelle, Grenié et al. manuscript that expresses environmental filtering, limiting similarity and hierarchical competition explicitely in function of species traits. This package provides all the code necessary to rerun the analyses of the manuscript.

Depends R (>= 3.5.0), ggplot2

LazyData true

License GPL (>= 2)

Encoding UTF-8

Imports dplyr, matrixStats, rlang, scales, tibble, tidyr, Weighted.Desc.Stat, magrittr

Suggests testthat, knitr, rmarkdown, stringr, furrr, forcats, spelling, tictoc

RoxygenNote 7.2.1

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

URL https://rekyt.github.io/fdcoexist

Repository https://rekyt.r-universe.dev

RemoteUrl https://github.com/Rekyt/fdcoexist

RemoteRef HEAD

RemoteSha 08d75af742c61d81d49b8699be8242b4dd67899e

Contents

alphaterm 2
bevHoltFct 3
check_trait_weights 3
compute_compet_distance 4

compute_hierarchical_compet 5
create_trait_weights 6
env_curve 6
extract_growth_rates 7
extract_mismatches 7
generate_cor_traits 8
generate_cor_traits_rand 9
mismatch 9
multigen 10
plot_patch 11
plot_rh 12
r_env 12
r_env_CT 13
scale_distance 14
sp_ab_gr 14
wtd_kurtosis 15
wtd_mean 15
wtd_skewness 16
wtd_var 16

Index **17**

`alphaterm` *Function definition for deterministic model run with global dispersal Compute alpha term in Beverton-Holt function*

Description

From a competition matrix (for the moment the distance between species traits) and a vector of abundances by species, return the alpha term in the Beverton-Holt equation. The order of species between the two should be the same as no checks are done. Typically the competition matrix is an euclidean trait distance matrix between species. The closer the species are the higher the combination. The term is computed as follow:

Usage

```
alphaterm(distance, Nts, A, B, di_thresh)
```

Arguments

`distance`	dissimilarity matrix between species
`Nts`	vector of abundances of species at time t
`A`	scalar for the inter-specific competition
`B`	scalar for the intra-specific competition
`di_thresh`	dissimilary threshold above which species are considered maximally dissimilar

Details

$$\alpha_i = \sum_{j=1, j \neq i}^{S} N_{t,j,x} \times (1 - \delta_{ij})$$

, where alpha_i is the competition term of species i; Ntjx the abundance of species j, at time t, in patch x and delta_ij the functional distance between species i and species j.

`bevHoltFct` *Beverton-Holt function*

Description

To simulate growth easily, use the Beverton-Holt equation. Which is:

Usage

```
bevHoltFct(R, N, alpha)
```

Arguments

`R`	a numeric vector of species growth rates
`N`	a numeric vector of species population sizes
`alpha`	the competition coefficient see `alphaterm()` for its computation

Details

$$N_{t+1,i,x} = \frac{R_{i,x} \times N_{t,i,x}}{1 + A \times \alpha}$$

where t is time, i is the species of interest and x the patch it occupies.

`check_trait_weights` *Check trait weights data.frame*

Description

This is an internal help function to check the trait weights data.frame (used in `multigen()`). The structure of the given trait weights is fixed: one column should be named `trait`` with the names of the traits in it. T` data.frame.

Usage

```
check_trait_weights(trait_weights, traits)
```

Arguments

`trait_weights`	data.frame with at least three columns equal to `trait` (giving the name of the concerned traits in `traits` df), `growth_weight` the relative weight of the trait in growth and `compet_weight` the relative weight of the trait in competition.
`traits`	a species-traits data.frame with species as rownames and traits as numeric columns with names matching `trait_weights` column `trait`

Value

nothing if data.frame passes the checks, stops early otherwise.

Examples

```
# Working trait weights data.frame
traits = data.frame(trait1 = 1, trait2 = 2, trait3 = 3)

weight_1 = data.frame(
   trait = c("trait1", "trait2", "trait3"),
   growth_weight    = c(0.5, 0.5, 0),
   compet_weight    = c(0,   0.5, 0.5),
   hierarchy_weight = c(0,   0,   0))

# Silent function
check_trait_weights(weight_1, traits)
## Not run:
# Not valid trait weights data.frame
not_valid = data.frame(trait = c("trait1", "trait2", "trait3"),
    growth_weight = c(0.5, 0.8, 0),
    compet_weight = c(0, 0.5, 0.9))

# Stop and error
check_trait_weights(not_valid, traits)

## End(Not run)
```

---

compute_compet_distance

Compute trait distance between species

Description

This function compute trait distance between species using a trait matrix and a trait weights data.frame. For all the traits with competition weights not equal to zero, it computes a weighted 'composite trait' that is then used to compute euclidean trait distance between species. Trait distance is first exponentiated then standardized between 0 and 1.

Usage

```
compute_compet_distance(trait_weights, traits, exponent = 1)
```

Arguments

`trait_weights`	data.frame with at least three columns equal to `trait` (giving the name of the concerned traits in `traits` df), `growth_weight` the relative weight of the trait in growth and `compet_weight` the relative weight of the trait in competition.
`traits`	a species-traits data.frame with species as rownames and traits as numeric columns with names matching `trait_weights` column `trait`
`exponent`	[`numeric(1)`] (default: 1) The exponent used before standardizing the distance

Value

an euclidean distance matrix (of type matrix)

`compute_hierarchical_compet`

Compute Hierarchical Competition coefficient at each time step

---

Description

Outputs a matrix of additional growth per patch per species given by hierarchical competition. The values are considered

Usage

```
compute_hierarchical_compet(
  composition_given_time_step,
  trait_values,
  trait_weights,
  H,
  exponent = 1
)
```

Arguments

`composition_given_time_step`	composition matrix at a given time step (a site-species matrix with sites in rows)
`trait_values`	a trait matrix
`trait_weights`	data.frame with at least three columns equal to `trait` (giving the name of the concerned traits in `traits` df), `growth_weight` the relative weight of the trait in growth and `compet_weight` the relative weight of the trait in competition.
`H`	the hierarchical competition scalar
`exponent`	exponent to use for hierarchical compet. distances

`create_trait_weights` *Generates a data.frame of trait weights*

Description

Only in the special case of 3 traits with 1 always driving growth and another one only driving competition

Usage

```
create_trait_weights(R, A, H, n_traits = 4)
```

Arguments

`R`	an integer value giving the contribution of trait in growth
`A`	an integer value giving the contribution of trait in limiting similarity
`H`	an integer value giving the contribution of trait in hierarchical competition
`n_traits`	number of traits considered in table

Examples

```
create_trait_weights(50, 50, 0)
```

`env_curve` *Species growth rate for a given trait and environment*

Description

Using traits that affect growth rate and specified environments, this function returns a numeric value of expected growth rates given the traits and the environmental value. The total growth rate is then the average of the growth rates computed with each trait.

Usage

```
env_curve(trait_values, env_value, trait_weights, k = 2, width = 0.5)
```

Arguments

`trait_values`	a numeric vector of species trait values
`env_value`	a single numeric value giving the environmental variable
`trait_weights`	data.frame with at least three columns equal to `trait` (giving the name of the concerned traits in `traits` df), `growth_weight` the relative weight of the trait in growth and `compet_weight` the relative weight of the trait in competition, both hierarchical and based on limiting similarity.
`k`	a scalar giving the maximum growth rate in optimal environment
`width`	a numeric for niche breadth, constant in gaussian function

Details

For the moment the environmental filter follows a Gaussian distribution:

$$R_{i,x} = k \times \exp(-\frac{(trait_i - env_x)^2}{2 \times width^2})$$

, where t_i is trait of species i, env_x the environmental value in patch x, k a scalar giving the maximal growth rate and width the environmental breadth of species.

`extract_growth_rates` *Extract different growth rates from fdcoexist simulation*

Description

Extract different growth rates from fdcoexist simulation

Usage

```
extract_growth_rates(simul, chosen_time = NULL)
```

Arguments

`simul`	Simulation object from `multigen()`
`chosen_time`	numeric timestep at which to extract growth rates

`extract_mismatches` *Extract species mismatches*

Description

Extract species mismatches

Usage

```
extract_mismatches(x, z)
```

Arguments

`x`	Simulation object from `multigen()`
`z`	Number of the species

generate_cor_traits *Generate correlated traits*

Description

This function generates a matrix of traits based on the given number of patches, species, the number of "additional" traits with a given correlation coefficient. The first trait is always uniform between 1 and the number of patches provided. Then the other traits are generated correlated to this first one. In the end all the traits are scaled between 1 and the number of patches.

Usage

```
generate_cor_traits(
  number_patches,
  number_species,
  number_other = 9,
  cor_coef = 0.7,
  min_value = 1
)
```

Arguments

`number_patches`	a numeric value giving the total number of patches
`number_species`	a numeric value giving the total number of species to simulate (number of rows in the trait tables)
`number_other`	a numeric value giving the number of additional traits to generate in addition to the trait correlated to the number of patches
`cor_coef`	a numeric value giving the correlation coefficient between the first trait and the other ones
`min_value`	a minimum trait value

Value

a matrix of traits with species in rows and traits in columns

Examples

```
traits <- generate_cor_traits(25, 100, 3, 0.3)
```

`generate_cor_traits_rand`
Generate random traits Compared to generate_cor_traits() introduce a little of variability in first trait as instead of being directly determined by the species number it adds little white noise to it and scale it to a minimum of 0 if negative and maximum of 25 if maximum value is over 25.

Description

Generate random traits Compared to generate_cor_traits() introduce a little of variability in first trait as instead of being directly determined by the species number it adds little white noise to it and scale it to a minimum of 0 if negative and maximum of 25 if maximum value is over 25.

Usage

```
generate_cor_traits_rand(
  number_patches,
  number_species,
  number_other = 9,
  cor_coef = 0.7,
  min_value = 1
)
```

Arguments

`number_patches`	Number of patches to consider
`number_species`	Number of species to which generate the traits
`number_other`	(default = 9) Number of other traits to generate
`cor_coef`	(default = 0.7) Correlation coefficient between first and other traits
`min_value`	absolute minimum trait value

`mismatch` *Plot mismatch per species between environmental optimum and max. abundance*

Description

Plot mismatch per species between environmental optimum and max. abundance

Usage

```
mismatch(simul, n_patches = 25, sp, time = 50, plot = TRUE)
```

Arguments

`simul`	results array of a given simulation (a `multigen()` output)
`n_patches`	an integer indicating the patch number
`sp`	integer for the number of species to consider
`time`	an integer indicating the number of time steps
`plot`	a boolean determining whether to plot or not the mismatch

---

`multigen` *Function to run the simulation*

Description

Using specified parameters this function run the simulation

Usage

```
multigen(
  traits,
  trait_weights,
  env,
  time,
  species,
  patches,
  composition,
  A = A,
  B = B,
  d,
  k,
  width,
  H,
  h_fun = "+",
  di_thresh = 24,
  lim_sim_exponent = 2,
  hierar_exponent = 0.5
)
```

Arguments

`traits`	a species-traits data.frame with species as rownames and traits as numeric columns with names matching `trait_weights` column `trait`
`trait_weights`	data.frame with at least three columns equal to `trait` (giving the name of the concerned traits in `traits` df), `growth_weight` the relative weight of the trait in growth and `compet_weight` the relative weight of the trait in competition.
`env`	a vector of environmental values

`time`	an integer giving the total number of generations
`species`	an integer giving the total number of species to simulate
`patches`	an integer giving the total number of patches to simulate
`composition`	the actual array containing species abundances per site over time, giving the initial populations of each species
`A`	the scalar of inter-specific competition coefficient (see `bevHoltFct()`)
`B`	the scalar for intra-specific competition coefficient by default B = A
`d`	a numeric value between 0 and 1 giving the percentage of dispersal occurring across all patches
`k`	a scalar giving the maximum growth rate in optimal environment
`width`	a numeric giving niche breadth of all species
`H`	a numeric for hierarchical competition such as H/k <= 1
`h_fun`	a function that describes how hierarchical is combined to environmental-based growth (default: `sum()`)
`di_thresh`	dissimilary threshold above which species are considered maximally dissimilar
`lim_sim_exponent`	exponent to use for limiting similarity distances
`hierar_exponent`	exponent to use for hierarchical compet. distances

`plot_patch` *Plot patch dynamics*

Description

Plot the dynamics of species in a single patch over time. Automatically assigns one color per species.

Usage

```
plot_patch(results, patch, time, equilibrium = FALSE)
```

Arguments

`results`	results array of a given simulation (a `multigen()` output)
`patch`	an integer indicating the patch number
`time`	an integer indicating the maximum timestep to look at
`equilibrium`	a boolean, if TRUE, displays a vertical line when equilibrium is reached

plot_rh — *Plot hierarchical growth of species in all the patches*

Description

Automatically assigns one color per species.

Usage

```
plot_rh(simul, n_patches, time)
```

Arguments

`simul`	results array of a given simulation (a `multigen()` output)
`n_patches`	an integer indicating the patch number
`time`	an integer indicating the number of time steps

r_env — *Plot environmental response curves of species in all the patches*

Description

Automatically assigns one color per species.

Usage

```
r_env(simul, n_patches, sp, plot = TRUE)
```

Arguments

`simul`	results array of a given simulation (a `multigen()` output)
`n_patches`	an integer indicating the patch number
`sp`	integer for the number of species to consider
`plot`	a boolean determining whether to plot or not the response curves

`r_env_CT` *Extract Species by Patch Growth Rates and Optimal Patches*

Description

Extract Species by Patch Growth Rates and Optimal Patches

Usage

```
r_env_CT(simul, n_patches, sp1, time)
```

Arguments

`simul`	a simulation output from `multigen()`
`n_patches`	the number of patches in the simulation
`sp1`	the number of species
`time`	the time slice at which e

Value

Outputs 4 data.frame of species by patch mismatches each with four columns:

- `env_all` -> Environmental Growth only
- `r_all` -> Environmental + Hierarchical competition Growth only
- `r_comp` -> Abundance
- `r_envab` -> Abundance based on environmental filtering only

The 4 data.frame are:

- `env` -> Patch
- `sp` -> Species
- `r_env` -> Observed statistic (environmental growth, total growth, abundance, environmental filtering abundance)
- `max_r_env` -> Patch of maximum observed statistic

`scale_distance` *Scale distance or matrix between 0 and 1*

Description

dist - min(dist) / (max(dist) - min(dist))

Usage

```
scale_distance(dist_matrix)
```

Arguments

`dist_matrix`	a matrix or a dist object

`sp_ab_gr` *Plot environmental response curves of species in all the patches*

Description

Automatically assigns one color per species.

Usage

```
sp_ab_gr(simul, n_patches, sp, time)
```

Arguments

`simul`	results array of a given simulation (a `multigen()` output)
`n_patches`	an integer indicating the patch number
`sp`	integer for the number of species to consider
`time`	integer determining the time step of interest

`wtd_kurtosis` *Weighted Kurtosis with na.rm*

Description

Compute weighted kurtosis using `Weighted.Desc.Stat::w.kurtosis` but add an option to remove `NA` values

Usage

```
wtd_kurtosis(x, w, na.rm = TRUE)
```

Arguments

`x`	values whose weighted mean is to be computed
`w`	weights of the same length as `x` to be used against `x`
`na.rm`	a logical values indicating whether `NA` values in both `x` and `w` should be stripped before computation

`wtd_mean` *Weighted mean that allows NA in values and weights*

Description

Weighted mean that allows NA in values and weights

Usage

```
wtd_mean(x, w, na.rm = TRUE)
```

Arguments

`x`	values whose weighted mean is to be computed
`w`	weights of the same length as `x` to be used against `x`
`na.rm`	a logical values indicating whether `NA` values in both `x` and `w` should be stripped before computation

wtd_skewness *Weighted Skewness with na.rm*

Description

Compute weighted Skewness using `Weighted.Desc.Stat::w.skewness` but add an option to remove `NA` values

Usage

```
wtd_skewness(x, w, na.rm = TRUE)
```

Arguments

`x`	values whose weighted mean is to be computed
`w`	weights of the same length as `x` to be used against `x`
`na.rm`	a logical values indicating whether `NA` values in both `x` and `w` should be stripped before computation

wtd_var *Weighted Variance*

Description

Weighted Variance

Usage

```
wtd_var(x, w, na.rm = TRUE)
```

Arguments

`x`	values whose weighted mean is to be computed
`w`	weights of the same length as `x` to be used against `x`
`na.rm`	a logical values indicating whether `NA` values in both `x` and `w` should be stripped before computation

Index

alphaterm, 2
alphaterm(), *3*

bevHoltFct, 3
bevHoltFct(), *11*

check_trait_weights, 3
compute_compet_distance, 4
compute_hierarchical_compet, 5
create_trait_weights, 6

env_curve, 6
extract_growth_rates, 7
extract_mismatches, 7

generate_cor_traits, 8
generate_cor_traits_rand, 9

mismatch, 9
multigen, 10
multigen(), *3*, *7*, *10–12*, *14*

plot_patch, 11
plot_rh, 12

r_env, 12
r_env_CT, 13

scale_distance, 14
sp_ab_gr, 14

Weighted.Desc.Stat::w.kurtosis, *15*
Weighted.Desc.Stat::w.skewness, *16*
wtd_kurtosis, 15
wtd_mean, 15
wtd_skewness, 16
wtd_var, 16