Package `miaSim`

April 16, 2024

**Type** Package  
**Version** 1.8.0  
**Title** Microbiome Data Simulation  
**Description** Microbiome time series simulation with generalized Lotka-Volterra model, Self-Organized Instability (SOI), and other models. Hubbell's Neutral model is used to determine the abundance matrix. The resulting abundance matrix is applied to (Tree)SummarizedExperiment objects.  
**License** Artistic-2.0 | file LICENSE  
**biocViews** Microbiome, Software, Sequencing, DNASeq, ATACSeq, Coverage, Network  
**Encoding** UTF-8  
**RoxygenNote** 7.2.3  
**Depends** TreeSummarizedExperiment  
**Imports** SummarizedExperiment, deSolve, stats, poweRlaw, MatrixGenerics, S4Vectors  
**Suggests** ape, cluster, foreach, doParallel, dplyr, GGally, ggplot2, igraph, network, reshape2, sna, vegan, rmarkdown, knitr, BiocStyle, testthat, mia, miaViz, colourvalues, philentropy  
**URL** [https://github.com/microbiome/miaSim](https://github.com/microbiome/miaSim)  
**BugReports** [https://github.com/microbiome/miaSim/issues](https://github.com/microbiome/miaSim/issues)  
**Roxygen** list(markdown = TRUE)  
**VignetteBuilder** knitr  
**git_url** [https://git.bioconductor.org/packages/miaSim](https://git.bioconductor.org/packages/miaSim)  
**git_branch** RELEASE_3_18  
**git_last_commit** 0e92339  
**git_last_commit_date** 2023-10-24  
**Repository** Bioconductor 3.18  
**Date/Publication** 2024-04-15
**.applyInterType**

Generate pairs of interactions according to interaction types

**Description**

A helper function to be used in combination with .getInteractions()

**Usage**

```
.applyInterType(I, pair, interType)
```
.estimateAFromSimulations

Arguments

I  Matrix: defining the interaction between each pair of species
pair Numeric: a vector with a length of 2, indicating the 2 focusing species in the process of applying the interaction types
interType Character: one of ‘mutualism’, ‘commensalism’, ‘parasitism’, ‘amensalism’, or ‘competition’. Defining the interaction type

Value

A matrix of interaction types with one pair changed

Description

generate matrix A from the comparisons between simulations with one absent species and a simulation with complete species (leave-one-out)

Usage

.estimateAFromSimulations(
    simulations,
    simulations2,
    n_instances = 1,
    t_end = NULL,
    scale_off_diagonal = 0.1,
    diagonal = -0.5,
    connectance = 0.2
)

Arguments

simulations A list of simulation(s) with complete species
simulations2 A list of simulation(s), each with one absent species
n_instances Integer: number of instances to generate (default: n_instances = 1)
t_end Numeric: end time of the simulation. If not identical with t_end in params_list, then it will overwrite t_end in each simulation (default: t_end = 1000)
scale_off_diagonal Numeric: scale of the off-diagonal elements compared to the diagonal. Same to the parameter in function randomA. (default: scale_off_diagonal = 0.1)
diagonal Values defining the strength of self-interactions. Input can be a number (will be applied to all species) or a vector of length n_species. Positive self-interaction values lead to exponential growth. Same to the parameter in function randomA. (default: diagonal = -0.5)
connectance Numeric frequency of inter-species interactions. i.e. proportion of non-zero off-diagonal terms. Should be in the interval $0 \leq \text{connectance} \leq 1$. Same to the parameter in function randomA. (default: connectance = 0.2)

Value

a matrix $A$ with dimensions $(n\_species \times n\_species)$ where $n\_species$ equals to the number of elements in simulations

Description

generate a vector of times when events is happening

Usage

.eventTimes(t\_events = NULL, t\_duration = NULL, t\_end = 1000, ...)

Arguments

t\_events, t\_duration
Numeric: vector of starting time and duration of the events

t\_end
Numeric: end time of the simulation

...: additional parameters to pass to simulationTimes, including t\_start, t\_step, and t\_store.

Value

A vector of time points in the simulation

Examples

tEvent <- .eventTimes(
    t\_events = c(10, 50, 100),
    t\_duration = c(1, 2, 3),
    t\_end = 100,
    t\_store = 100,
    t\_step = 1
)

**.getInteractions**

Generate interactions according to five types of interactions and their weights

**Description**

Generate interactions according to five types of interactions and their weights

**Usage**

`.getInteractions(n_species, weights, connectance)`

**Arguments**

- `n_species` Integer: defining the dimension of matrix of interaction
- `weights` Numeric: defining the weights of mutualism, commensalism, parasitism, amensalism, and competition in all interspecies interactions.
- `connectance` Numeric: defining the density of the interaction network. Ranging from 0 to 1

**Value**

A matrix of interactions with all interactions changed according to the weights and connectance.

**.isPosInt**

check whether a number is a positive integer

**Description**

check whether a number is a positive integer

**Usage**

`.isPosInt(x, tol = .Machine$double.eps^0.5)`

**Arguments**

- `x` Numeric number to test
- `tol` Numeric tolerance of detection

**Value**

A logical value: whether the number is a positive integer.
.rdirichlet

Generate dirichlet random deviates

Description

Generate dirichlet random deviates

Usage

.rdirichlet(n, alpha)

Arguments

n
  Number of random vectors to generate.
alpha
  Vector containing shape parameters.

Value

A vector containing the Dirichlet density

Examples

dirichletExample <- .rdirichlet(1, c(1, 2, 3))

.replaceByZero

Replace one element with zero in a list.

Description

If the list contains m elements, then lengths of each element must be m, too. This function is intended to generate a list of x0 (the initial community) with one missing species, to prepare the parameter simulations_compare in estimateAFromSimulations.

Usage

.replaceByZero(input_list)

Arguments

input_list
  A list containing m elements, and lengths of each element must be m, too.

Value

A list of same dimension as input_list, but with 0 at specific positions in the elements of the list.
.simulationTimes

Generate simulation times and the indices of time points to return in simulation functions.

Description

Generate simulation times and the indices of time points to return in simulation functions.

Usage

.simulationTimes(t_start = 0, t_end = 1000, t_step = 0.1, t_store = 1000)

Arguments

t_start Numeric scalar indicating the initial time of the simulation. (default: t_start = 0)
t_end Numeric scalar indicating the final time of the simulation (default: t_end = 1000)
t_step Numeric scalar indicating the interval between simulation steps (default: t_step = 0.1)
t_store Integer scalar indicating the number of evenly distributed time points to keep (default: t_store = 100)

Value

lists containing simulation times (t_sys) and the indices to keep.

Examples

Time <- .simulationTimes(
  t_start = 0, t_end = 100, t_step = 0.5,
  t_store = 100
)
DefaultTime <- .simulationTimes(t_end = 1000)

powerlawA

Interaction matrix with Power-Law network adjacency matrix

Description

N is the an Interspecific Interaction matrix with values drawn from a normal distribution H the interaction strength heterogeneity drawn from a power-law distribution with the parameter alpha, and G the adjacency matrix of with out-degree that reflects the heterogeneity of the powerlaw. A scaling factor s may be used to constrain the values of the interaction matrix to be within a desired range. Diagonal elements of A are defined by the parameter d.
Usage

powerlawA(n_species, alpha = 3, stdev = 1, s = 0.1, d = -1, symmetric = FALSE)

Arguments

- **n_species**: integer number of species
- **alpha**: numeric power-law distribution parameter. Should be > 1. (default: alpha = 3.0) Larger values will give lower interaction strength heterogeneity, whereas values closer to 1 give strong heterogeneity in interaction strengths between the species. In other words, values of alpha close to 1 will give Strongly Interacting Species (SIS).
- **stdev**: numeric standard deviation parameter of the normal distribution with mean 0 from which the elements of the nominal interspecific interaction matrix N are drawn. (default: stdev = 1)
- **s**: numeric scaling parameter with which the final global interaction matrix A is multiplied. (default: s = 0.1)
- **d**: numeric diagonal values, indicating self-interactions (use negative values for stability). (default: s = 1.0)
- **symmetric**: logical scalar returning a symmetric interaction matrix (default: symmetric=FALSE)

Value

The interaction matrix A with dimensions (n_species x n_species)

References


Examples

```r
# Low interaction heterogeneity
A_low <- powerlawA(n_species = 10, alpha = 3)
# Strong interaction heterogeneity
A_strong <- powerlawA(n_species = 10, alpha = 1.01)
```

randomA

Generate random interaction matrix for GLV model

Description

Generates a random interaction matrix for Generalized Lotka-Volterra (GLV) model.
randomA

Usage

randomA(
  n_species,
  names_species = NULL,
  diagonal = -0.5,
  connectance = 0.2,
  scale_off_diagonal = 0.1,
  mutualism = 1,
  commensalism = 1,
  parasitism = 1,
  amensalism = 1,
  competition = 1,
  interactions = NULL,
  symmetric = FALSE,
  list_A = NULL
)

Arguments

n_species  Integer: number of species
names_species  Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
diagonal  Values defining the strength of self-interactions. Input can be a number (will be applied to all species) or a vector of length n_species. Positive self-interaction values lead to exponential growth. (default: diagonal = -0.5)
connectance  Numeric frequency of inter-species interactions. i.e. proportion of non-zero off-diagonal terms. Should be in the interval 0 <= connectance <= 1. (default: connectance = 0.2)
scale_off_diagonal  Numeric: scale of the off-diagonal elements compared to the diagonal. (default: scale_off_diagonal = 0.1)
mutualism  Numeric: relative proportion of interactions terms consistent with mutualism (positive <-> positive) (default: mutualism = 1)
commensalism  Numeric: relative proportion of interactions terms consistent with commensalism (positive <-> neutral) (default: commensalism = 1)
parasitism  Numeric: relative proportion of interactions terms consistent with parasitism (positive <-> negative) (default: parasitism = 1)
amensalism  Numeric: relative proportion of interactions terms consistent with amensalism (neutral <-> negative) (default: amensalism = 1)
competition  Numeric: relative proportion of interactions terms consistent with competition (negative <-> negative) (default: competition = 1)
interactions  Numeric: values of the n_species^2 pairwise interaction strengths. Diagonal terms will be replaced by the 'diagonal' parameter If NULL, interactions are drawn from runif(n_species^2, min=0, max=abs(diagonal)). Negative values are first converted to positive then the signs are defined by the relative
weights of the biological interactions (i.e. mutualism, commensalism, parasitism, amensalism, competition) (default: interactions = NULL)

**symmetric**
Logical: whether the strength of mutualistic and competitive interactions are symmetric. This is implemented by overwrite a half of the matrix, so the proportions of interactions might deviate from expectations. (default: symmetric=FALSE)

**list_A**
List: a list of matrices generated by randomA. Used to support different groups of interactions. If NULL (by default), no group is considered. Otherwise the given list of matrices will overwrite values around the diagonal. (default: list_A = NULL)

**Value**
randomA returns a matrix A with dimensions (n_species x n_species)

**Examples**
```r
dense_A <- randomA(
  n_species = 10,
  scale_off_diagonal = 1,
  diagonal = -1.0,
  connectance = 0.9
)

sparse_A <- randomA(
  n_species = 10,
  diagonal = -1.0,
  connectance = 0.09
)

user_interactions <- rbeta(n = 10^2, .5, .5)
user_A <- randomA(n_species = 10, interactions = user_interactions)

competitive_A <- randomA(
  n_species = 10,
  mutualism = 0,
  commensalism = 0,
  parasitism = 0,
  amensalism = 0,
  competition = 1,
  connectance = 1,
  scale_off_diagonal = 1
)

parasitism_A <- randomA(
  n_species = 10,
  mutualism = 0,
  commensalism = 0,
  parasitism = 1,
  amensalism = 0,
  competition = 0,
  connectance = 1,
)```
randomE

```r
  scale_off_diagonal = 1,
  symmetric = TRUE
)

list_A <- list(dense_A, sparse_A, competitive_A, parasitism_A)
groupA <- randomA(n_species = 40, list_A = list_A)
```

**randomE**  Generate random efficiency matrix

---

**Description**  Generate random efficiency matrix for consumer resource model from Dirichlet distribution, where positive efficiencies indicate the consumption of resources, whilst negatives indicate that the species would produce the resource.

**Usage**  
randomE(
  n_species,
  n_resources,
  names_species = NULL,
  names_resources = NULL,
  mean_consumption = n_resources/4,
  mean_production = n_resources/6,
  maintenance = 0.5,
  trophic_levels = NULL,
  trophic_preferences = NULL,
  exact = FALSE
)

**Arguments**

- **n_species**  Integer: number of species
- **n_resources**  Integer: number of resources
- **names_species**  Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
- **names_resources**  Character: names of resources. If NULL, paste0("res", seq_len(n_resources)) is used.
- **mean_consumption**  Numeric: mean number of resources consumed by each species drawn from a poisson distribution (default: mean_consumption = n_resources/4)
- **mean_production**  Numeric: mean number of resources produced by each species drawn from a poisson distribution (default: mean_production = n_resources/6)
maintenance Numeric: proportion of resources that cannot be converted into products between 0–1 the proportion of resources used to maintain the living of microorganisms. 0 means all the resources will be used for the reproduction of microorganisms, and 1 means all the resources would be used to maintain the living of organisms and no resources would be left for their growth(reproduction). (default: maintenance = 0.5)

trophic_levels Integer: number of species in microbial trophic levels. If NULL, by default, microbial trophic levels would not be considered. (default: trophic_levels = NULL)

trophic_preferences List: preferred resources and productions of each trophic level. Positive values indicate the consumption of resources, whilst negatives indicate that the species would produce the resource.

exact Logical: whether to set the number of consumption/production to be exact as mean_consumption/mean_production or to set them using a Poisson distribution. (default: exact = FALSE) If length(trophic_preferences) is smaller than length(trophic_levels), then NULL values would be appended to lower trophic levels. If NULL, by default, the consumption preference will be defined randomly. (default: trophic_preferences = NULL)

Value

randomE returns a matrix E with dimensions (n_species x n_resources), and each row represents a species.

Examples

# example with minimum parameters
ExampleEfficiencyMatrix <- randomE(n_species = 5, n_resources = 12)

# examples with specific parameters
ExampleEfficiencyMatrix <- randomE(
  n_species = 3, n_resources = 6,
  names_species = letters[1:3],
  names_resources = paste0("res", LETTERS[1:6]),
  mean_consumption = 3, mean_production = 1
)

ExampleEfficiencyMatrix <- randomE(
  n_species = 3, n_resources = 6,
  maintenance = 0.4
)

ExampleEfficiencyMatrix <- randomE(
  n_species = 3, n_resources = 6,
  mean_consumption = 3, mean_production = 1, maintenance = 0.4
)

# examples with microbial trophic levels
ExampleEfficiencyMatrix <- randomE(
  n_species = 10, n_resources = 15,
  trophic_levels = c(6, 3, 1),
)
simulateConsumerResource

Consumer-resource model simulation

Description

Simulates time series with the consumer-resource model.

Usage

```r
simulateConsumerResource(
  n_species,
  n_resources,
  names_species = NULL,
  names_resources = NULL,
  E = NULL,
  x0 = NULL,
  resources = NULL,
  resources_dilution = NULL,
  growth_rates = NULL,
  monod_constant = NULL,
  sigma_drift = 0.001,
  sigma_epoch = 0.1,
  sigma_external = 0.3,
  sigma_migration = 0.01,
  epoch_p = 0.001,
  t_external_events = NULL,
  t_external_durations = NULL,
  stochastic = FALSE,
  migration_p = 0.01,
  metacommunity_probability = NULL,
)```

simulateConsumerResource

error_variance = 0,
norm = FALSE,
t_end = 1000,
trophic_priority = NULL,
inflow_rate = 0,
outflow_rate = 0,
volume = 1000,
...
)

Arguments

n_species  Integer: number of species
n_resources  Integer: number of resources
names_species  Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
names_resources  Character: names of resources. If NULL, paste0("res", seq_len(n_resources)) is used.
E  matrix: matrix of efficiency. A matrix defining the efficiency of resource-to-biomass conversion (positive values) and the relative conversion of metabolic by-products (negative values). If NULL, randomE(n_species, n_resources) is used. (default: E = NULL)
x0  Numeric: initial abundances of simulated species. If NULL, runif(n = n_species, min = 0.1, max = 10) is used. (default: x0 = NULL)
resources  Numeric: initial concentrations of resources. If NULL, runif(n = n_resources, min = 1, max = 100) is used. (default: resources = NULL)
resources_dilution  Numeric: concentrations of resources in the continuous inflow (applicable when inflow_rate > 0). If NULL, resources is used. (default: resources_dilution = NULL)
growth_rates  Numeric: vector of maximum growth rates(mu) of species. If NULL, rep(1, n_species) is used. (default: growth_rates = NULL)
monod_constant  matrix: the constant of additive monod growth of n_species consuming n_resources. If NULL, matrix(rgamma(n = n_species*n_resources, shape = 50*max(resources), rate = 1), nrow = n_species) is used. (default: monod_constant = NULL)
sigma_drift  Numeric: standard deviation of a normally distributed noise applied in each time step (t_step) (default: sigma_drift = 0.001)
sigma_epoch  Numeric: standard deviation of a normally distributed noise applied to random periods of the community composition with frequency defined by the epoch_p parameter (default: sigma_epoch = 0.1)
sigma_external  Numeric: standard deviation of a normally distributed noise applied to user-defined external events/disturbances (default: sigma_external = 0.3)
sigma_migration  Numeric: standard deviation of a normally distributed variable that defines the intensity of migration at each time step (t_step) (default: sigma_migration = 0.01)
epoch_p  Numeric: the probability/frequency of random periodic changes introduced to the community composition (default: epoch_p = 0.001)

t_external_events  Numeric: the starting time points of defined external events that introduce random changes to the community composition (default: t_external_events = NULL)

t_external_durations  Numeric: respective duration of the external events that are defined in the 't_external_events' (times) and sigma_external (std). (default: t_external_durations = NULL)

stochastic  Logical: whether to introduce noise in the simulation. If False, sigma_drift, sigma_epoch, and sigma_external are ignored. (default: stochastic = FALSE)

migration_p  Numeric: the probability/frequency of migration from a metacommunity. (default: migration_p = 0.01)

metacommunity_probability  Numeric: Normalized probability distribution of the likelihood that species from the metacommunity can enter the community during the simulation. If NULL, rdirichlet(1, alpha = rep(1,n_species)) is used. (default: metacommunity_probability = NULL)

t_end  Numeric: the end time of the simulationTimes, defining the modeled time length of the community. (default: t_end = 1000)

val

an TreeSummarizedExperiment class object
Examples

```r
n_species <- 2
n_resources <- 4
tse <- simulateConsumerResource(
  n_species = n_species,
  n_resources = n_resources
)

# example with user-defined values (names_species, names_resources, E, x0,
# resources, growth_rates, error_variance, t_end, t_step)
ExampleE <- randomE(
  n_species = n_species, n_resources = n_resources,
  mean_consumption = 3, mean_production = 1, maintenance = 0.4
)
ExampleResources <- rep(100, n_resources)
tse1 <- simulateConsumerResource(
  n_species = n_species,
  n_resources = n_resources, names_species = letters[seq_len(n_species)],
  names_resources = paste0("res", LETTERS[seq_len(n_resources)]), E = ExampleE,
  x0 = rep(0.001, n_species), resources = ExampleResources,
  growth_rates = runif(n_species),
  error_variance = 0.01,
  t_end = 5000,
  t_step = 1
)

# example with trophic levels
n_species <- 10
n_resources <- 15
ExampleEfficiencyMatrix <- randomE(
  n_species = 10, n_resources = 15,
  trophic_levels = c(6, 3, 1),
  trophic_preferences = list(
    c(rep(1, 5), rep(-1, 5), rep(0, 5)),
    c(rep(0, 5), rep(1, 5), rep(-1, 5)),
    c(rep(0, 10), rep(1, 5))
  )
)
ExampleResources <- c(rep(500, 5), rep(200, 5), rep(50, 5))
tse2 <- simulateConsumerResource(
  n_species = n_species,
  n_resources = n_resources,
  names_species = letters[1:n_species],
  names_resources = paste0("res", LETTERS[1:n_resources] ),
  E = ExampleEfficiencyMatrix,
  x0 = rep(0.001, n_species),
  resources = ExampleResources,
  growth_rates = rep(1, n_species),
)```

simulateGLV

Generalized Lotka-Volterra (gLV) simulation

Description

Simulates time series with the generalized Lotka-Volterra model.

Usage

simulateGLV(
  n_species,
  names_species = NULL,
  A = NULL,
  x0 = NULL,
  growth_rates = NULL,
  sigma_drift = 0.001,
  sigma_epoch = 0.1,
  sigma_external = 0.3,
sigma_migration = 0.01,
epoch_p = 0.001,
t_external_events = NULL,
t_external_durations = NULL,
stochastic = TRUE,
migration_p = 0.01,
metacommunity_probability = NULL,
error_variance = 0,
norm = FALSE,
t_end = 1000,
...
)

Arguments

n_species  Integer: number of species
names_species  Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
A  matrix: interaction matrix defining the positive and negative interactions between n_species. If NULL, randomA(n_species) is used. (default: A = NULL)
x0  Numeric: initial abundances of simulated species. If NULL, runif(n = n_species, min = 0, max = 1) is used. (default: x0 = NULL)
growth_rates  Numeric: growth rates of simulated species. If NULL, runif(n = n_species, min = 0, max = 1) is used. (default: growth_rates = NULL)
sigma_drift  Numeric: standard deviation of a normally distributed noise applied in each time step (t_step) (default: sigma_drift = 0.001)
sigma_epoch  Numeric: standard deviation of a normally distributed noise applied to random periods of the community composition with frequency defined by the epoch_p parameter (default: sigma_epoch = 0.1)
sigma_external  Numeric: standard deviation of a normally distributed noise applied to user-defined external events/disturbances (default: sigma_external = 0.3)
sigma_migration  Numeric: standard deviation of a normally distributed variable that defines the intensity of migration at each time step (t_step) (default: sigma_migration = 0.01)
epoch_p  Numeric: the probability/frequency of random periodic changes introduced to the community composition (default: epoch_p = 0.001)
t_external_events  Numeric: the starting time points of defined external events that introduce random changes to the community composition (default: t_external_events = NULL)
t_external_durations  Numeric: respective duration of the external events that are defined in the 't_external_events' (times) and sigma_external (std). (default: t_external_durations = NULL)
stochastic  Logical: whether to introduce noise in the simulation. If False, sigma_drift, sigma_epoch, and sigma_external are ignored. (default: stochastic = FALSE)
simulateGLV

**migration_p**  Numeric: the probability/frequency of migration from a metacommunity. (default: migration_p = 0.01)

**metacommunity_probability**  Numeric: Normalized probability distribution of the likelihood that species from the metacommunity can enter the community during the simulation. If NULL, rdirichlet(1, alpha = rep(1, n_species)) is used. (default: metacommunity_probability = NULL)

**error_variance**  Numeric: the variance of measurement error. By default it equals to 0, indicating that the result won’t contain any measurement error. This value should be non-negative. (default: error_variance = 0)

**norm**  Logical: whether the time series should be returned with the abundances as proportions (norm = TRUE) or the raw counts (default: norm = FALSE) (default: norm = FALSE)

**t_end**  Numeric: the end time of the simulationTimes, defining the modeled time length of the community. (default: t_end = 1000)

...  additional parameters, see *utils* to know more.

**Details**

Simulates a community time series using the generalized Lotka-Volterra model, defined as \( \frac{dx}{dt} = x(b+Ax) \), where \( x \) is the vector of species abundances, \( \text{diag}(x) \) is a diagonal matrix with the diagonal values set to \( x \). \( A \) is the interaction matrix and \( b \) is the vector of growth rates.

**Value**

simulateGLV returns a TreeSummarizedExperiment class object

**Examples**

```r
# generate a random interaction matrix
ExampleA <- randomA(n_species = 4, diagonal = -1)

# run the model with default values (only stochastic migration considered)
tse <- simulateGLV(n_species = 4, A = ExampleA)

# run the model with two external disturbances at time points 240 and 480 # with durations equal to 1 (10 time steps when t_step by default is 0.1).
ExampleGLV <- simulateGLV(
  n_species = 4, A = ExampleA,
  t_external_events = c(0, 240, 480), t_external_durations = c(0, 1, 1)
)

# run the model with no perturbation nor migration
set.seed(42)
tse1 <- simulateGLV(
  n_species = 4, A = ExampleA, stochastic = FALSE,
  sigma_migration = 0
)
```
# run the model with no perturbation nor migration but with measurement error
set.seed(42)
tse2 <- simulateGLV(
    n_species = 4, A = ExampleA, stochastic = FALSE,
    error_variance = 0.001, sigma_migration = 0
)

simulateHubbell  
*Hubbell's neutral model simulation*

**Description**

Neutral species abundances simulation according to the Hubbell model.

**Usage**

```r
simulateHubbell(
    n_species,
    M,
    carrying_capacity = 1000,
    k_events = 10,
    migration_p = 0.02,
    t_skip = 0,
    t_end,
    norm = FALSE
)
```

**Arguments**

- `n_species`: integer amount of different species initially in the local community
- `M`: integer amount of different species in the metacommunity, including those of the local community
- `carrying_capacity`: integer value of fixed amount of individuals in the local community (default: `carrying_capacity = 1000`)
- `k_events`: integer value of fixed amount of deaths of local community individuals in each generation (default: `k_events = 10`)
- `migration_p`: numeric immigration rate: the probability that a death in the local community is replaced by a migrant of the metacommunity rather than by the birth of a local community member (default: `migration_p = 0.02`)
- `t_skip`: integer number of generations that should not be included in the outputted species abundance matrix. (default: `t_skip = 0`)
- `t_end`: integer number of simulations to be simulated
- `norm`: logical scalar choosing whether the time series should be returned with the abundances as proportions (`norm = TRUE`) or the raw counts (default: `norm = FALSE`)
Value

simulateHubbell returns a TreeSummarizedExperiment class object

References


Examples

tse <- simulateHubbell(
    n_species = 8, M = 10, carrying_capacity = 1000, k_events = 50,
    migration_p = 0.02, t_end = 100
)

simulateHubbellRates  Hubbell's neutral model simulation applied to time series

Description

Neutral species abundances simulation according to the Hubbell model. This model shows that losses in society can be replaced either by the birth of individuals or by immigration depending on their probabilities. The specific time between the events of birth or migration is calculated and time effect is considered to determine the next event.

Usage

simulateHubbellRates(
    n_species = NULL,
    x0 = NULL,
    names_species = NULL,
    migration_p = 0.01,
    metacommunity_probability = NULL,
    k_events = 1,
    growth_rates = NULL,
    error_variance = 0,
    norm = FALSE,
    t_end = 1000,
    ...
)

Arguments

n_species  Integer: number of species

x0  Numeric: initial species composition. If NULL, rep(100, n_species) is used.
names_species  Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)

migration_p  Numeric: the probability/frequency of migration from a metacommunity. (default: migration_p = 0.01)

metacommunity_probability  Numeric: Normalized probability distribution of the likelihood that species from the metacommunity can enter the community during the simulation. If NULL, rdirichlet(1, alpha = rep(1,n_species)) is used. (default: metacommunity_probability = NULL)

k_events  Integer: number of events to simulate before updating the sampling distributions. (default: k_events = 1)

growth_rates  Numeric: maximum growth rates(mu) of species. If NULL, rep(1, n_species) is used. (default: growth_rates = NULL)

error_variance  Numeric: the variance of measurement error. By default it equals to 0, indicating that the result won't contain any measurement error. This value should be non-negative. (default: error_variance = 0)

norm  Logical: whether the time series should be returned with the abundances as proportions (norm = TRUE) or the raw counts (default: norm = FALSE) (default: norm = FALSE)

t_end  Numeric: the end time of the simulationTimes, defining the modeled time length of the community. (default: t_end = 1000)

...  additional parameters, see utils to know more.

Value

simulateHubbellRates returns a TreeSummarizedExperiment class object

References


Examples

set.seed(42)
tse <- simulateHubbellRates(n_species = 5)

miaViz::plotSeries(tse, x = "time")

# no migration, all stochastic birth and death
set.seed(42)
tse1 <- simulateHubbellRates(n_species = 5, migration_p = 0)

# all migration, no stochastic birth and death
set.seed(42)
tse2 <- simulateHubbellRates(
  n_species = 5,
  migration_p = 1,
simulateRicker

Generate time series with the Ricker model

Description

The Ricker model is a discrete version of the generalized Lotka-Volterra model and is implemented here as proposed by Fisher and Mehta in PLoS ONE 2014.

Usage

simulateRicker(
  n_species,
  A,
  names_species = NULL,
  x0 = runif(n_species),
  carrying_capacities = runif(n_species),
  error_variance = 0.05,
  explosion_bound = 10^8,
  t_end = 1000,
  norm = FALSE,
  ...)

# all migration, no stochastic birth and death, but with measurement errors
set.seed(42)
tse3 <- simulateHubbellRates(
  n_species = 5,
  migration_p = 1,
  metacommunity_probability = c(0.1, 0.15, 0.2, 0.25, 0.3),
  t_end = 20,
  t_store = 200,
  error_variance = 100
)

# model with specified inputs
set.seed(42)
tse4 <- simulateHubbellRates(
  n_species = 5,
  migration_p = 0.1,
  metacommunity_probability = c(0.1, 0.15, 0.2, 0.25, 0.3),
  t_end = 200,
  t_store = 1000,
  k_events = 5,
  growth_rates = c(1.1, 1.05, 1, 0.95, 0.9)
)
**simulateSOI**

**Self-Organised Instability model (SOI) simulation**

**Description**


**Arguments**

- `n_species` Integer: number of species
- `A` interaction matrix
- `names_species` Character: names of species. If NULL, `paste0("sp", seq_len(n_species))` is used. (default: `names_species = NULL`)
- `x0` Numeric: initial abundances of simulated species. If NULL, `runif(n = n_species, min = 0, max = 1)` is used.
- `carrying_capacities` numeric carrying capacities. If NULL, `runif(n = n_species, min = 0, max = 1)` is used.
- `error_variance` Numeric: the variance of measurement error. By default it equals to 0, indicating that the result won’t contain any measurement error. This value should be non-negative. (default: `error_variance = 0.05`)
- `explosion_bound` numeric value of boundary for explosion (default: `explosion_bound = 10^8`)
- `t_end` integer number of simulations to be simulated
- `norm` logical scalar returning normalised abundances (proportions in each generation) (default: `norm = FALSE`)
- `...` additional parameters, see `utils` to know more.

**Value**

`simulateRicker` returns a `TreeSummarizedExperiment` class object

**References**


**Examples**

```r
A <- powerlawA(10, alpha = 1.01)
tse <- simulateRicker(n_species = 10, A, t_end = 100)
```
Usage

simulateSOI(
  n_species,
  x0 = NULL,
  names_species = NULL,
  carrying_capacity = 1000,
  A = NULL,
  k_events = 5,
  t_end = 1000,
  metacommunity_probability = runif(n_species, min = 0.1, max = 0.8),
  death_rates = runif(n_species, min = 0.01, max = 0.08),
  norm = FALSE
)

Arguments

n_species Integer: number of species
x0 a vector of initial community abundances If (default: x0 = NULL), based on migration rates
names_species Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
carrying_capacity integer community size, number of available sites (individuals)
A matrix: interaction matrix defining the positive and negative interactions between n_species. If NULL, powerlawA(n_species) is used. (default: A = NULL)
k_events integer number of transition events that are allowed to take place during one leap. (default: k_events = 5). Higher values reduce runtime, but also accuracy of the simulation.
t_end Numeric: the end time of the simulation, defining the modeled time length of the community. (default: t_end = 1000)
metacommunity_probability Numeric: Normalized probability distribution of the likelihood that species from the metacommunity can enter the community during the simulation. By default, runif(n_species, min = 0.1, max = 0.8) is used. (default: metacommunity_probability = runif(n_species, min = 0.1, max = 0.8))
death_rates Numeric: death rates of each species. By default, runif(n_species, min = 0.01, max = 0.08) is used. (default: death_rates = runif(n_species, min = 0.01, max = 0.08))
norm logical scalar indicating whether the time series should be returned with the abundances as proportions (norm = TRUE) or the raw counts (default: norm = FALSE)

Value

simulateSOI returns a TreeSummarizedExperiment class object
Examples

```r
# Generate interaction matrix
A <- miASim::powerlawA(10, alpha = 1.2)
# Simulate data from the SOI model
tse <- simulateSOI(
  n_species = 10, carrying_capacity = 1000, A = A,
  k_events = 5, x0 = NULL, t_end = 150, norm = TRUE
)
```

**simulateStochasticLogistic**

*Stochastic Logistic simulation*

Description

Simulates time series with the (stochastic) logistic model

Usage

```r
simulateStochasticLogistic(
  n_species, 
  names_species = NULL, 
  growth_rates = NULL, 
  carrying_capacities = NULL, 
  death_rates = NULL, 
  x0 = NULL, 
  sigma_drift = 0.001, 
  sigma_epoch = 0.1, 
  sigma_external = 0.3, 
  sigma_migration = 0.01, 
  epoch_p = 0.001, 
  t_external_events = NULL, 
  t_external_durations = NULL, 
  migration_p = 0.01, 
  metacommunity_probability = NULL, 
  stochastic = TRUE, 
  error_variance = 0, 
  norm = FALSE, 
  t_end = 1000, 
  ...
)
```

Arguments

- **n_species**  | Integer: number of species
names_species Character: names of species. If NULL, paste0("sp", seq_len(n_species)) is used. (default: names_species = NULL)
growth_rates Numeric: growth rates of simulated species. If NULL, runif(n = n_species, min = 0.1, max = 0.2) is used. (default: growth_rates = NULL)
carrying_capacities Numeric: The max population of species supported in the community. If NULL, runif(n = n_species, min = 1000, max = 2000) is used. (default: carrying_capacities = NULL)
death_rates Numeric: death rates of each species. If NULL, runif(n = n_species, min = 0.0005, max = 0.0025) is used. (default: death_rates = NULL)
x0 Numeric: initial abundances of simulated species. If NULL, runif(n = n_species, min = 0.1, max = 10) is used. (default: x0 = NULL)
sigma_drift Numeric: standard deviation of a normally distributed noise applied in each time step (t_step) (default: sigma_drift = 0.001)
sigma_epoch Numeric: standard deviation of a normally distributed noise applied to random periods of the community composition with frequency defined by the epoch_p parameter (default: sigma_epoch = 0.1)
sigma_external Numeric: standard deviation of a normally distributed noise applied to user-defined external events/disturbances (default: sigma_external = 0.3)
sigma_migration Numeric: standard deviation of a normally distributed variable that defines the intensity of migration at each time step (t_step) (default: sigma_migration = 0.01)
epoch_p Numeric: the probability/frequency of random periodic changes introduced to the community composition (default: epoch_p = 0.001)
t_external_events Numeric: the starting time points of defined external events that introduce random changes to the community composition (default: t_external_events = NULL)
t_external_durations Numeric: respective duration of the external events that are defined in the 't_external_events' (times) and sigma_external (std). (default: t_external_durations = NULL)
migration_p Numeric: the probability/frequency of migration from a metacommunity. (default: migration_p = 0.01)
metacommunity_probability Numeric: Normalized probability distribution of the likelihood that species from the metacommunity can enter the community during the simulation. If NULL, rdirichlet(1, alpha = rep(1,n_species)) is used. (default: metacommunity_probability = NULL)
stochastic Logical: whether to introduce noise in the simulation. If False, sigma_drift, sigma_epoch, and sigma_external are ignored. (default: stochastic = TRUE)
error_variance Numeric: the variance of measurement error. By default it equals to 0, indicating that the result won’t contain any measurement error. This value should be non-negative. (default: error_variance = 0)
**simulateStochasticLogistic**

- **norm**
  Logical: whether the time series should be returned with the abundances as proportions (norm = TRUE) or the raw counts (default: norm = FALSE) (default: norm = FALSE)

- **t_end**
  Numeric: the end time of the simulationTimes, defining the modeled time length of the community. (default: t_end = 1000)

- **...**
  additional parameters, see *utils* to know more.

**Details**

The change rate of the species was defined as $dx/dt = b*x^*(1-(x/k))*rN - dr*x$, where $b$ is the vector of growth rates, $x$ is the vector of initial species abundances, $k$ is the vector of maximum carrying capacities, $rN$ is a random number ranged from 0 to 1 which changes in each time step, $dr$ is the vector of constant death rates. Also, the vectors of initial dead species abundances can be set. The number of species will be set to 0 if the dead species abundances surpass the alive species abundances.

**Value**

simulateStochasticLogistic returns a TreeSummarizedExperiment class object

**Examples**

```r
# Example of logistic model without stochasticity, death rates, or external # disturbances
set.seed(42)
tse <- simulateStochasticLogistic(
  n_species = 5,
  stochastic = FALSE, death_rates = rep(0, 5)
)

# Adding a death rate
set.seed(42)
tse1 <- simulateStochasticLogistic(
  n_species = 5,
  stochastic = FALSE, death_rates = rep(0.01, 5)
)

# Example of stochastic logistic model with measurement error
set.seed(42)
tse2 <- simulateStochasticLogistic(
  n_species = 5,
  error_variance = 1000
)

# example with all the initial parameters defined by the user
set.seed(42)
tse3 <- simulateStochasticLogistic(
  n_species = 2,
  names_species = c("species1", "species2"),
  growth_rates = c(0.2, 0.1),
  carrying_capacities = c(1000, 2000),
  ...)
```
death_rates = c(0.001, 0.0015),
x0 = c(3, 0.1),
sigma_drift = 0.001,
sigma_epoch = 0.3,
sigma_external = 0.5,
sigma_migration = 0.002,
epoch_p = 0.001,
t_external_events = c(100, 200, 300),
t_external_durations = c(0.1, 0.2, 0.3),
migration_p = 0.01,
metacommunity_probability = miaSim::rdirichlet(1, alpha = rep(1, 2)),
stochastic = TRUE,
error_variance = 0,
norm = FALSE, # TRUE,
t_end = 400,
t_start = 0, t_step = 0.01,
t_store = 1500
)

Index

* internal
  .simulationTimes, 7
  .applyInterType, 2
  .estimateAFromSimulations, 3
  .eventTimes, 4
  .getInteractions, 5
  .isPosInt, 5
  .rdirichlet, 6
  .replaceByZero, 6
  .simulationTimes, 7

powerlawA, 7

randomA, 8
randomE, 11

simulateConsumerResource, 13
simulateGLV, 17
simulateHubbell, 20
simulateHubbellRates, 21
simulateHubbellRates,numeric-method
  (simulateHubbellRates), 21
simulateHubbellRates-numeric
  (simulateHubbellRates), 21
simulateRicker, 23
simulateS0I, 24
simulateStochasticLogistic, 26

utils, 15, 19, 22, 24, 28