Package ‘cxr’

October 28, 2019

Type Package

Title A Toolbox for Modelling Species Coexistence in R

Version 0.1

Date 2019-10-28

Description Recent developments in modern coexistence theory has advanced
our understanding on how species are able to persist and co-occur
with other species at varying abundances. However, applying this
mathematical framework to empirical data is still challenging,
precluding a larger adoption of the theoretical tools developed
by empiricists. This package provides a complete toolbox for modelling
competitive effects between species, calculate fitness and
niche differences, and calculate coexistence regions.
The functions are flexible and can include covariates
and different fitting algorithms can be used.

License MIT + file LICENCE

Depends R (>= 3.5)

Imports stats, nlme

RoxygenNote 6.1.1

Suggests ggplot2, tidyr, dplyr, magrittr, knitr, stringr, rmarkdown,
testthat (>= 0.8.0), hydroPSO, GenSA, DEoptimR, nloptr

VignetteBuilder knitr

NeedsCompilation no

Author David Garcia-Callejas [aut, cre]
(<https://orcid.org/0000-0001-6982-476X>),
Ignasi Bartomeus [aut] (<https://orcid.org/0000-0001-7893-4389>),
Oscar Godoy [aut] (<https://orcid.org/0000-0003-4988-6626>),
Maxime Lancelot [ctb]

Maintainer David Garcia-Callejas <david.garcia.callejas@gmail.com>

R topics documented:

`abundance` *Abundance measurements*

Description

A dataset containing abundances for each plant species, where each species was sampled at its developmental peak.

Usage

```
data(abundance)
```

Format

A data frame with 5832 rows and 7 variables

Details

- year: year
- month: month of sampling
- day: day of sampling
- plot: plot
- subplot: subplot code
- species: plant species
- individuals: number of individuals

Note

For details, see Lanuza et al. 2018 Ecology Letters.

`AvgFitnessRatio` *Average fitness differences between a pair of species*

Description

Calculates the product of (1) the demographic ratio, and (2) the competitive response ratio between two species according to their vital rates and competition coefficients. First species in the parameters is numerator (j in eq. 4 of Godoy et al. 2014). If germ.rate and survival.rate are provided, it calculates the ratio according to eq. 4 of Godoy et al. (2014), for annual plants. Otherwise, if only lambda is provided, it returns the general version of the demographic ratio, where nu = lambda

Usage

```
AvgFitnessRatio(lambda, germ.rate = NULL, survival.rate = NULL,
  pair.matrix)
```

Arguments

`lambda`	vector of length 2, per capita fecundity of the species in the absence of competition
`germ.rate`	optional vector of length 2, germination rate of the two species
`survival.rate`	optional vector of length 2, annual survival of ungerminated seed in the soil
`pair.matrix`	2x2 matrix, competition coefficients between the two species and intraspecific terms

Value

list with three numeric values, giving 1) the demographic ratio, 2) the competitive response ratio, and 3) the average fitness ratio between the two species

`competition` *Competition measurements*

Description

A dataset containing fitness and neighbours for each plant individual

Usage

```
data(competition)
```

Format

A data frame with 42948 rows and 11 variables

Details

- year: year
- month: month
- day: day
- plot: plot
- subplot: subplot code
- focal: focal plant species identity
- individual_ID: unique identifier for each individual focal plant
- fruit: total fruits produced by the focal individual
- seed: total seeds produced by the focal individual
- competitor: competitor identity
- number: number of competitors

Note

For details, see Lanuza et al. 2018 Ecology Letters.

`cxr` *cxr*

Description

Tools and functions for evaluating multi-species coexistence.

`cxr_er_bootstrap` *standard error estimates for effect and response parameters*

Description

Computes bootstrap standard errors for a given effect/response function

Usage

```
cxr_er_bootstrap(effect.response.model, optim.method, sp.data, init.par,
  lower.bounds, upper.bounds, covariates, optimize.lambda, lambda.vector,
  nsamples)
```

Arguments

`effect.response.model`	effect/response function
`optim.method`	optimization method to use. One of the following: "optim_NM","optim_L-BFGS-B","nloptr_CRS2_LM", "nloptr_ISRES","nloptr_DIRECT_L_RAND","GenSA","hydroPSO","D
`sp.data`	dataframe with all the necessary information in long format. It should have the following columns: - site: character ID - focal: character ID of the focal species. Any number of focal species is allowed, but the number of focal species must match the number of initial parameters (one lambda, e, and r per species). - fitness: numeric, a fitness metric - competitor: character, ID of a competitor for that observation. The set of competitors must be, for now, the same as the set of focal species. - number: number of neighbouring/competitor individuals from the associated species. Observations without competitors of a given species must be explicit, i.e. setting number to zero.
`init.par`	1d vector of initial parameters
`lower.bounds`	1d vector of lower bounds
`upper.bounds`	1d vector of upper bounds
`covariates`	dataframe/matrix with observations in rows and covariates in columns. Each cell is the value of a covariate from an observation.
`optimize.lambda`	boolean, whether to optimize the values of lambda or not.
`lambda.vector`	in case lambda is not to be optimized, fixed values for it.
`nsamples`	how many bootstrap samples to compute.

Value

1d vector, the standard error of each parameter in init.par `cxr_init_params` *Join parameters in a 1d vector*

Description

Generate a 1d vector from a series of parameters in a certain order. It also returns the same vector for lower and upper bounds. This function is intended to work with parameters for a single species (i.e. a single lambda value, etc). Note that lambda.cov and alpha.cov must be consistent with num.covariates.

Usage

```
cxr_init_params(init.lambda = NULL, init.sigma = 0,
  init.alpha = NULL, init.lambda.cov = NULL, init.alpha.cov = NULL,
  lower.lambda = 1, upper.lambda = 1e+05, lower.sigma = 1e-05,
  upper.sigma = 1e+05, lower.alpha = 1e-05, upper.alpha = 1e+05,
  lower.lambda.cov = 1e-05, upper.lambda.cov = 1e+05,
  lower.alpha.cov = 1e-05, upper.alpha.cov = 1e+05, num.competitors,
  num.covariates)
```

Arguments

`init.lambda`	numeric, lambda
`init.sigma`	numeric, sigma
`init.alpha`	1d vector, interaction coefficients over the species
`init.lambda.cov`	1d vector, initial values for lambda.cov
`init.alpha.cov`	1d vector, initial values for alpha.cov
`lower.lambda`	lower bound for lambda
`upper.lambda`	upper bound for lambda
`lower.sigma`	lower bound for sigma
`upper.sigma`	upper bound for sigma
`lower.alpha`	lower bound for alpha
`upper.alpha`	upper bound for alpha
`lower.lambda.cov`	lower bound for lambda.cov
`upper.lambda.cov`	upper bound for lambda.cov
`lower.alpha.cov`	lower bound for alpha.cov
`upper.alpha.cov`	upper bound for alpha.cov
`num.competitors`	number of competitors
`num.covariates`	number of covariates

Value

list with three 1d vectors, ready for passing to the optim methods, consistent with the functions model_BH1-5

---

cxr_pm_bootstrap *Standard error estimates for model parameters*

Description

Computes bootstrap standard errors for a given population dynamics model.

Usage

```
cxr_pm_bootstrap(fitness.model, optim.method, param.list, fixed.terms,
  log.fitness, init.par, lower.bounds, upper.bounds, focal.comp.matrix,
  focal.covariates, nsamples)
```

Arguments

`fitness.model`	function returning a single value to minimize, given a set of parameters and a fitness metric
`optim.method`	optimization method to use. One of the following: "optim_NM","optim_L-BFGS-B","nloptr_CRS2_LM", "nloptr_ISRES","nloptr_DIRECT_L_RAND","GenSA","hydroPSO","D
`param.list`	string vector giving the parameters that are to be optimized for the fitness model.
`fixed.terms`	string vector giving the parameters that are NOT optimized for the fitness model.
`log.fitness`	1d vector, log of the fitness metric for every observation
`init.par`	1d vector of initial parameters
`lower.bounds`	1d vector of lower bounds
`upper.bounds`	1d vector of upper bounds
`focal.comp.matrix`	matrix with observations in rows and neighbours in columns. Each cell is the number of neighbours of a given species in a given observation.
`focal.covariates`	optional matrix with observations in rows and covariates in columns. Each cell is the value of a covariate in a given observation.
`nsamples`	how many bootstrap samples to compute.

Value

1d vector, the standard error of each parameter in init.par

`cxr_retrieve_params` *Retrieve parameters from the vector returned by the optimization procedures*

Description

Retrieve parameters from the vector returned by the optimization procedures

Usage

```
cxr_retrieve_params(optim.params, param.list, alpha.length,
  alpha.cov.length, num.competitors, num.covariates)
```

Arguments

`optim.params`	1d vector, the result of an optimization method
`param.list`	character vector, which parameters are present. Possible elements are "lambda", "lambda.cov", "alpha", "alpha.cov".
`alpha.length`	if alpha is to be retrieved, its length
`alpha.cov.length`	if alpha.cov is to be retrieved, its length
`num.competitors`	how many competitor species
`num.covariates`	how many covariates

Value

list with elements "lambda", "alpha", "lambda.cov", "alpha.cov", "sigma". If one of these elements is not present, returns NULL.

`er_optim` *Estimate competition effects and responses for a set of species*

Description

This function is similar in spirit to pm_optim, in that it optimizes a set of parameters via maximum likelihood. See vignette 'Obtain competitive responses and effects' for an example of its use.

Usage

```
er_optim(lambda.vector, e.vector, r.vector, lambda.cov = NULL,
  e.cov = NULL, r.cov = NULL, sigma, lower.lambda = 0,
  upper.lambda = 1000, lower.e, upper.e, lower.r, upper.r,
  lower.lambda.cov = NULL, upper.lambda.cov = NULL,
  lower.e.cov = NULL, upper.e.cov = NULL, lower.r.cov = NULL,
  upper.r.cov = NULL, lower.sigma, upper.sigma, effect.response.model,
  optim.method, sp.data, covariates = NULL, optimize.lambda = FALSE,
  generate.errors = FALSE, bootstrap.samples = 0)
```

Arguments

`lambda.vector`	1d vector of lambda estimates/initial values (depending on whether lambda values are optimized or not)
`e.vector`	1d vector of competitive effect initial values
`r.vector`	1d vector of competitive response initial values
`lambda.cov`	numeric matrix of num.sp x num.covariates, effect of every covariate on species' lambda. Discarded if covariates are not passed.
`e.cov`	numeric matrix of num.sp x num.covariates, effect of every covariate on species' competitive effect. Discarded if covariates are not passed.
`r.cov`	numeric matrix of num.sp x num.covariates, effect of every covariate on species' competitive response. Discarded if covariates are not passed.
`sigma`	initial value for variation estimate.
`lower.lambda`	lower bound for lambda, in case it is optimized. Either length 1 or same length as lambda.vector
`upper.lambda`	upper bound for lambda, in case it is optimized. Either length 1 or same length as lambda.vector
`lower.e`	lower bound for competitive effects. Either length 1 or same length as e.vector
`upper.e`	upper bound for competitive effects. Either length 1 or same length as e.vector
`lower.r`	lower bound for competitive responses. Either length 1 or same length as r.vector
`upper.r`	upper bound for competitive responses. Either length 1 or same length as r.vector
`lower.lambda.cov`	lower bound for covariate effects on lambda. Either length 1 or same length as lambda.cov. Discarded if covariates are not passed.
`upper.lambda.cov`	upper bound for covariate effects on lambda. Either length 1 or same length as lambda.cov. Discarded if covariates are not passed.
`lower.e.cov`	lower bound for covariate effects on e. Either length 1 or same length as e.cov. Discarded if covariates are not passed.
`upper.e.cov`	upper bound for covariate effects on e. Either length 1 or same length as e.cov. Discarded if covariates are not passed.
`lower.r.cov`	lower bound for covariate effects on r. Either length 1 or same length as r.cov. Discarded if covariates are not passed.

`upper.r.cov`	upper bound for covariate effects on r. Either length 1 or same length as r.cov. Discarded if covariates are not passed.
`lower.sigma`	lower bound for sigma. Length 1.
`upper.sigma`	upper bound for sigma. Length 1.
`effect.response.model`	function returning a value to optimize over, e.g. maximum likelihood
`optim.method`	optimization method to use. One of the following: "optim_NM","optim_L-BFGS-B","nloptr_CRS2_LM", "nloptr_ISRES","nloptr_DIRECT_L_RAND","GenSA","hydroPSO","D
`sp.data`	dataframe with all the necessary information in long format. It should have the following columns: - site: character ID - focal: character ID of the focal species. Any number of focal species is allowed, but the number of focal species must match the number of initial parameters (one lambda, e, and r per species). - fitness: numeric, a fitness metric - competitor: character, ID of a competitor for that observation. The set of competitors must be, for now, the same as the set of focal species. - number: number of neighbouring/competitor individuals from the associated species. Observations without competitors of a given species must be explicit, i.e. setting number to zero.
`covariates`	optional matrix/dataframe with as many rows as the sp.data dataframe, and covariates in columns.
`optimize.lambda`	boolean, whether we want to optimize lambda values or not.
`generate.errors`	boolean, whether to compute bootstrap errors for the fitted parameters. Note that, depending on the data and optimization method, this may be computationally expensive.
`bootstrap.samples`	how many bootstrap samples to compute.

Value

list with estimated species values for e, r, lambda (optional), and if covariates are given, the effects of covariates on lambda, r, and e.

`GenerateTestData` *Generate simulated competition data*

Description

Generate simulated competition data

Usage

```
GenerateTestData(focal.sp = 1, num.sp = 2, num.cov = 2,
  num.obs = 10, fitness.model = 1, focal.lambda, alpha,
  alpha.cov = NULL, lambda.cov = NULL)
```

Arguments

`focal.sp`	number of focal species
`num.sp`	total number of species, including focal ones
`num.cov`	number of covariates
`num.obs`	number of observations/sites
`fitness.model`	scalar representing the model to generate data from, from BH1 to BH5 in increasing levels of complexity.
`focal.lambda`	1d vector with lambdas of the focal sp
`alpha`	interaction matrix, num.sp x num.sp
`alpha.cov`	list of dimension num.cov. Each component of the list is a matrix of different dimensions depending on `fitness.model`. If fitness.model is 4, each component should be a single value, giving the effect of each covariate over every interaction; if fitness.model is 5, each component should be a matrix num.sp x num.sp, giving the effect of the covariate in question over each element of the interaction matrix.
`lambda.cov`	matrix of num.sp x num.cov giving the effect of each covariate over the fecundity (lambda) of each species

Value

dataset with a fitness metric calculated for each focal species and observation, according to the fitness model selected

---

`model_abundBH1` *Project abundance of individuals according to the Beverton-Holt fist model*

Description

Project abundance of individuals according to the Beverton-Holt fist model

Usage

```
model_abundBH1(sp.par, init.abund, cov.values, alpha.matrix,
  lambda.cov.matrix, alpha.cov.matrix, return.seeds = TRUE)
```

Arguments

`sp.par`	dataframe with species in rows, and the following columns: lambda: fecundity term germ.rate: seed germination rate survival.rate: annual survival of ungerminated seed
`init.abund`	number of individuals at time t
`cov.values`	Not used in BH_abundances_1
`alpha.matrix`	Not used in BH_1

`lambda.cov.matrix`	Not used in BH_abundances_1
`alpha.cov.matrix`	Not used in BH_abundances_1
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

1d vector with number of individuals of each species at time t+1

`model_abundBH2` *Project abundance of individuals according to the Beverton-Holt second model*

Description

Project abundance of individuals according to the Beverton-Holt second model

Usage

```
model_abundBH2(sp.par, init.abund, cov.values, alpha.matrix,
  lambda.cov.matrix, alpha.cov.matrix, return.seeds = TRUE)
```

Arguments

`sp.par`	dataframe with species in rows, and the following columns: lambda: fecundity term germ.rate: seed germination rate survival.rate: annual survival of ungerminated seed
`init.abund`	number of individuals at time t
`cov.values`	Not used in model_abundBH2
`alpha.matrix`	competition value, same for all interactions
`lambda.cov.matrix`	Not used in model_abundBH2
`alpha.cov.matrix`	Not used in model_abundBH2
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

1d vector with number of individuals of each species at time t+1

`model_abundBH3` *Project abundance of individuals according to the Beverton-Holt third model*

Description

Project abundance of individuals according to the Beverton-Holt third model

Usage

```
model_abundBH3(sp.par, init.abund, cov.values, alpha.matrix,
  lambda.cov.matrix, alpha.cov.matrix, return.seeds = TRUE)
```

Arguments

`sp.par`	dataframe with species in rows, and the following columns: lambda: fecundity term germ.rate: seed germination rate survival.rate: annual survival of ungerminated seed
`init.abund`	number of individuals at time t
`cov.values`	Not used in model_abundBH3
`alpha.matrix`	competition matrix
`lambda.cov.matrix`	Not used in model_abundBH3
`alpha.cov.matrix`	Not used in model_abundBH3
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

1d vector with number of individuals of each species at time t+1

`model_abundBH4` *Project abundance of individuals according to the Beverton-Holt fourth model*

Description

Project abundance of individuals according to the Beverton-Holt fourth model

Usage

```
model_abundBH4(sp.par, init.abund, cov.values, alpha.matrix,
  lambda.cov.matrix, alpha.cov.matrix, return.seeds = TRUE)
```

Arguments

`sp.par`	dataframe with species in rows, and the following columns: lambda: fecundity term germ.rate: seed germination rate survival.rate: annual survival of ungerminated seed
`init.abund`	number of individuals at time t
`cov.values`	1d vector with values of each covariate
`alpha.matrix`	competition matrix
`lambda.cov.matrix`	matrix of num.sp x num.cov giving the effect of each covariate over the fecundity (lambda) of each species
`alpha.cov.matrix`	list of dimension number of covariates. Each component of the list is a single value, giving the effect of the covariate in question over the interaction matrix.
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

1d vector with number of individuals of each species at time t+1

`model_abundBH5` *Project abundance of individuals according to the Beverton-Holt fifth model*

Description

Project abundance of individuals according to the Beverton-Holt fifth model

Usage

```
model_abundBH5(sp.par, init.abund, cov.values, alpha.matrix,
  lambda.cov.matrix, alpha.cov.matrix, return.seeds = TRUE)
```

Arguments

`sp.par`	dataframe with species in rows, and the following columns: lambda: fecundity term germ.rate: seed germination rate survival.rate: annual survival of ungerminated seed
`init.abund`	number of individuals at time t
`cov.values`	1d vector with values of each covariate
`alpha.matrix`	competition matrix
`lambda.cov.matrix`	matrix of num.sp x num.cov giving the effect of each covariate over the fecundity (lambda) of each species

`alpha.cov.matrix`	list of dimension number of covariates. Each component of the list is a matrix of num.sp x num.sp, giving the effect of the covariate in question over the interaction matrix.
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

1d vector with number of individuals of each species at time t+1

`model_BH1` *Title Beverton-Holt fecundity, first model*

Description

These functions return the negative log-likelihood of the data given the model and parameters. model_BH1 is $F_i = \lambda_i$

Usage

```
model_BH1(par, param.list = NULL, log.fitness,
  focal.comp.matrix = NULL, num.covariates = NULL,
  num.competitors = NULL, focal.covariates = NULL,
  fixed.terms = NULL)
```

Arguments

`par`	vector containing lambda of focal sp and sigma value
`param.list`	not used in model_BH1
`log.fitness`	log of fitness value
`focal.comp.matrix`	not used in model_BH1
`num.covariates`	not used in model_BH1
`num.competitors`	not used in model_BH1
`focal.covariates`	not used in model_BH1
`fixed.terms`	not used in model_BH1

Value

log-likelihood value

`model_BH2` *Title Beverton-Holt fecundity, second model*

Description

These functions return the negative log-likelihood of the data given the model and parameters. model_BH2 is $F_i = \frac{\lambda_i}{1+\alpha \sum_j N_j}$

Usage

```
model_BH2(par, param.list = c("lambda", "alpha"), log.fitness,
  focal.comp.matrix, num.covariates = NULL, num.competitors = NULL,
  focal.covariates = NULL, fixed.terms = NULL)
```

Arguments

`par`	vector of variable length, with the following order: first, lambda of focal sp; second alpha, single interaction coefficient; last, sigma value. If any element is not to be optimized, it must not be present in this vector, but rather in the "fixed.terms" list
`param.list`	string listing parameters to optimize. Possible elements are `lambda`, `lambda.cov`, `alpha`, `alpha.cov`.
`log.fitness`	log of fitness value
`focal.comp.matrix`	dataframe with as many rows as observations, and one column for each competitor sp. Values of the dataframe are number of competitors of each sp per observation.
`num.covariates`	not used in model_BH2
`num.competitors`	not used in model_BH2
`focal.covariates`	not used in model_BH2.
`fixed.terms`	list with elements `lambda`, `lambda.cov`, `alpha`, `alpha.cov`. It contains parameters not to be optimized. Each element of the list must be of its appropriate length. Note that adding an element in "param.list" will force the function to look for it in `par`, and will not consider it here. In this model, `lambda.cov` and `alpha.cov` are not considered.

Value

log-likelihood value

`model_BH3` *Title Beverton-Holt fecundity, third model*

Description

These functions return the negative log-likelihood of the data given the model and parameters. model_BH3 is $F_i = \frac{\lambda_i}{1+\sum_j \alpha_{ij} N_j}$

Usage

```
model_BH3(par, param.list = c("lambda", "alpha"), log.fitness,
  focal.comp.matrix, num.covariates = NULL, num.competitors = NULL,
  focal.covariates = NULL, fixed.terms = NULL)
```

Arguments

`par`	vector of variable length, with the following order: first, lambda of focal sp; second alpha, interaction coefficients with every species; last, sigma value. If any element is not to be optimized, it must not be present in this vector, but rather in the `fixed.terms` list
`param.list`	string listing parameters to optimize. Possible elements are `lambda`, `lambda.cov`, `alpha`, `alpha.cov`.
`log.fitness`	log of fitness value
`focal.comp.matrix`	dataframe with as many rows as observations, and one column for each competitor sp. Values of the dataframe are number of competitors of each sp per observation.
`num.covariates`	not used in model_BH3
`num.competitors`	number of competitor species
`focal.covariates`	not used in model_BH3.
`fixed.terms`	list with elements `lambda`, `lambda.cov`, `alpha`, `alpha.cov`. It contains parameters not to be optimized. Each element of the list must be of its appropriate length. Note that adding an element in "param.list" will force the function to look for it in `par`, and will not consider it here. In this model, `lambda.cov` and `alpha.cov` are not considered.

Value

log-likelihood value

model_BH4 *Title Beverton-Holt fecundity, fourth model*

Description

These functions return the negative log-likelihood of the data given the model and parameters. model_BH4 is $F_i = \frac{\lambda_i + \sum_k b_k \Theta_{i,k}}{1 + \sum_j (\alpha_{ij} + \sum_k b_k \phi_k) N_j}$

Usage

```
model_BH4(par, param.list, log.fitness, focal.comp.matrix, num.covariates,
  num.competitors, focal.covariates, fixed.terms)
```

Arguments

`par`	vector of variable length, with the following order: first, lambda of focal sp; second lambda.cov, effects of every covariate on lambda; third alpha, interaction coefficients with every species; fourth alpha.cov, effects of every covariate on alpha values (single effect for each covariate); last, sigma value. If any element is not to be optimized, it must not be present in this vector, but rather in the `fixed.terms` list
`param.list`	string listing parameters to optimize. Possible elements are `lambda`, `lambda.cov`, `alpha`, `alpha.cov`.
`log.fitness`	log of fitness value
`focal.comp.matrix`	dataframe with as many rows as observations, and one column for each competitor sp. Values of the dataframe are number of competitors of each sp per observation.
`num.covariates`	number of covariates
`num.competitors`	number of competitor species
`focal.covariates`	dataframe/matrix with as many rows as observationes, and one column for each covariate. Values of the dataframe are covariate values for every observation.
`fixed.terms`	list with elements `lambda`, `lambda.cov`, `alpha`, `alpha.cov`. It contains parameters not to be optimized. Each element of the list must be of its appropriate length. Note that adding an element in "param.list" will force the function to look for it in `par`, and will not consider it here. In this model, `lambda.cov` and `alpha.cov` are not considered.

Value

log-likelihood value

model_BH5 — *Title Beverton-Holt fecundity, fifth model*

Description

These functions return the negative log-likelihood of the data given the model and parameters. model_BH5 is $F_i = \frac{\lambda_i + \sum_k b_k \Theta_{i,k}}{1 + \sum_j (\alpha_{ij} + \sum_k b_k \phi_{i,j,k}) N_j}$

Usage

```
model_BH5(par, param.list, log.fitness, focal.comp.matrix, num.covariates,
  num.competitors, focal.covariates, fixed.terms)
```

Arguments

`par`	vector of variable length, with the following order: first, lambda of focal sp; second lambda.cov, effects of every covariate on lambda; third alpha, interaction coefficients with every species; fourth alpha.cov, effects of every covariate on alpha values (varying effect of every covariate over every interaction coefficient); last, sigma value. If any element is not to be optimized, it must not be present in this vector, but rather in the `fixed.terms` list
`param.list`	string listing parameters to optimize. Possible elements are `lambda`, `lambda.cov`, `alpha`, `alpha.cov`.
`log.fitness`	log of fitness value
`focal.comp.matrix`	dataframe with as many rows as observations, and one column for each competitor sp. Values of the dataframe are number of competitors of each sp per observation.
`num.covariates`	number of covariates
`num.competitors`	number of competitor species
`focal.covariates`	dataframe with as many rows as observationes, and one column for each covariate. Values of the dataframe are covariate values for every observation.
`fixed.terms`	list with elements `lambda`, `lambda.cov`, `alpha`, `alpha.cov`. It contains parameters not to be optimized. Each element of the list must be of its appropriate length. Note that adding an element in "param.list" will force the function to look for it in `par`, and will not consider it here. In this model, `lambda.cov` and `alpha.cov` are not considered.

Value

log-likelihood value

`model_ER` *Estimation of competitive effects and responses*

Description

Calculates the log-likelihood of a Beverton-Holt model parameterized with given values with respect to a fitness metric. The function for calculating fecundity given effect and response values is taken from Godoy et al. (2014). Note that, as e and r are not pair-specific, all species parameters are fit in the same function. In this version, lambda values are fixed.

Usage

```
model_ER(init.par, lambda, target_all, density_all, log.fitness,
  covariates = NULL)
```

Arguments

`init.par`	1d vector of initial parameters: r values, e values, and single sigma term. If covariates are given, this vector must include also lambda.cov, response.cov, and effect.cov terms, after the e values and before the sigma term.
`lambda`	1d vector of lambda values
`target_all`	matrix giving which species is calculated with which values. See `er_optim`
`density_all`	matrix giving the densities of each species at each observation. See `er_optim`
`log.fitness`	log of the fitness metric
`covariates`	if present, it is a dataframe/matrix with as many rows as observationes, and one column for each covariate. Values are covariate values for every observation.

Value

single numeric value giving the sum of negative log-likelihoods

`model_ER_lambda` *Estimation of competitive effects and responses*

Description

Calculates the log-likelihood of a Beverton-Holt model parameterized with given values with respect to a fitness metric. The function for calculating fecundity given effect and response values is taken from Godoy et al. (2014). Note that, as e is not pair-specific, all species parameters are fit in the same function. In this version, lambda values are also fit.

Usage

```
model_ER_lambda(init.par, target_all, density_all, log.fitness,
  covariates = NULL)
```

Arguments

`init.par`	1d vector of initial parameters: lambda values, r values, e values, and single sigma term. If covariates are given, this vector must include also lambda.cov, response.cov, and effect.cov terms, after the e values and before the sigma term.
`target_all`	matrix giving which species is calculated with which values. See `er_optim`
`density_all`	matrix giving the densities of each species at each observation. See `er_optim`
`log.fitness`	log of the fitness metric
`covariates`	if present, it is a dataframe/matrix with as many rows as observationes, and one column for each covariate. Values are covariate values for every observation.

Value

single numeric value giving the sum of negative log-likelihoods

---

`NicheOverlap` *Niche overlap between two species*

Description

quoting Godoy et al. (2014): reflects the average degree to which species limit individuals of their own species relative to competitors. Low niche overlap causes species to have greater per capita growth rates when rare than when common. If species limit individuals of their own species and their competitors equally, then niche overlap is 1, and coexistence is not possible unless species are otherwise identical. At the other extreme, if species have no interspecific effects, then niche overlap is 0.

Usage

```
NicheOverlap(pair.matrix)
```

Arguments

`pair.matrix`	2x2 matrix with competition coefficients between the two species, and intraspecific terms

Value

niche overlap value, in the range 0-1.

`param_estimates` *Population model parameters*

Description

A dataset containing estimated parameters for the competition dataset The dataset is generated by code similar to that of the vignette `Multi-species parameter optimization`. It is structured as a nested list with three levels, of the form: param_estimates[[species]][[fecundity model]][[optimization method]] In the provided dataset, we include models 1 to 5, estimated with optimization method "optim_L-BFGS-B" For each combination of species, model, and method, all model parameters are given. If a parameter is not estimated (e.g. upper/lower errors), the value stored is NA.

Usage

```
data(param_estimates)
```

Format

A nested list with three levels

Details

- lambda: per germinant fecundity
- lambda.lower.error:
- lambda.upper.error:
- sigma:
- alpha: effect of every competitor species on the focal species
- alpha.lower.error:
- alpha.upper.error:
- lambda.cov: effect of every covariate on lambda
- lambda.cov.lower.error:
- lambda.cov.upper.error:
- alpha.cov: effect of every covariate on alpha
- alpha.cov.lower.error:
- alpha.cov.upper.error:
- log.likelihood: log-likelihood of the fitted parameters and model

Note

For details, see vignette `Multi-species parameter optimization` `pm_optim` *General optimization for population models*

Description

Wrapper for optimization procedures. It accepts a population dynamics model, defined as a function, and a series of parameters. It returns the optimal value for the parameters given a fitness metric and an optimization method. Optionally, bootstrap confidence intervals can also be computed.

Usage

```
pm_optim(fitness.model, optim.method, param.list, log.fitness,
  init.lambda = NULL, lower.lambda = 1, upper.lambda = 1e+05,
  init.sigma = NULL, lower.sigma = 1e-10, upper.sigma = 1e+05,
  init.alpha = 1e-04, lower.alpha = 0, upper.alpha = 1e+05,
  init.lambda.cov = 0.001, lower.lambda.cov = 1e-04,
  upper.lambda.cov = 1e+05, init.alpha.cov = 0.001,
  lower.alpha.cov = 1e-04, upper.alpha.cov = 1e+05, focal.comp.matrix,
  focal.covariates = NULL, generate.errors = FALSE,
  bootstrap.samples = 0, verbose = FALSE)
```

Arguments

`fitness.model`	function giving the population dynamics model. Any functional form is allowed, but the model must be constrained to free parameters `lambda` (fecundity of each sp in absence of competition), `alpha` (interaction coefficients), `lambda.cov` (effect of covariates on lambda), `alpha.cov` (effect of covariates on alpha)
`optim.method`	optimization method to use. One of the following: "optim_NM","optim_L-BFGS-B","nloptr_CRS2_LM", "nloptr_ISRES","nloptr_DIRECT_L_RAND","GenSA","hydroPSO","D
`param.list`	string vector giving the parameters that are to be optimized for the fitness model (to choose among "lambda", "alpha", "lambda.cov", and "alpha.cov").
`log.fitness`	1d vector, log of the fitness metric for every observation
`init.lambda`	1d vector, initial value of lambda
`lower.lambda`	lower bound for lambda
`upper.lambda`	upper bound for lambda
`init.sigma`	initial value for sigma (standard deviation)
`lower.sigma`	lower bound for sigma
`upper.sigma`	upper bound for sigma
`init.alpha`	initial value for the alpha vector/matrix
`lower.alpha`	lower bound for alpha
`upper.alpha`	upper bound for alpha
`init.lambda.cov`	initial value for the lambda.cov matrix. Discarded if no covariates are given.

`lower.lambda.cov`	lower bound for lambda.cov
`upper.lambda.cov`	upper bound for lambda.cov
`init.alpha.cov`	initial value for the alpha.cov matrix. Discarded if no covariates are given.
`lower.alpha.cov`	lower bound for alpha.cov
`upper.alpha.cov`	upper bound for alpha.cov
`focal.comp.matrix`	matrix with observations in rows and neighbours in columns. Each cell is the number of neighbours of a given species in a given observation.
`focal.covariates`	optional matrix with observations in rows and covariates in columns. Each cell is the value of a covariate in a given observation.
`generate.errors`	boolean, whether to compute bootstrap errors for the fitted parameters. Note that, depending on the data, model, and optimization method, this may be computationally expensive.
`bootstrap.samples`	how many bootstrap samples to compute.
`verbose`	work in progress

Value

list with the fitted parameters, and the loglikelihood of the fit. If a parameter is taken as a constant, the list will return the original value given.

`pollinators` *Pollinators measurments*

Description

A dataset containing the overall pollinator visitation over the season for each plant species.

Usage

```
data(pollinators)
```

Format

A data frame with x rows and 9 variables

Details

- plot: plot number
- subplot: subplot code
- plant_species: plant species
- year: year of measurement
- bees: total bee visits recieved
- flies: total flies visits recieved
- beetles: total beetles visits recieved
- butterflies: total butterflies visits recieved
- other: total other visits recieved

Note

For details, see Lanuza et al. 2018 Ecology Letters.

`PredictAbundances` *Project abundances according to specified models and parameters*

Description

Project abundances according to specified models and parameters

Usage

```
PredictAbundances(par, timesteps, abundance.model, return.seeds = TRUE)
```

Arguments

`par`	list with the following components: <ul><li>dataframe sp.par, with the parameters to be passed to the predictive model</li><li>dataframe initial.values, with fields "site","species","abundance"</li><li>covariates, either 0 if there are no covariates, or a dataframe with fields "site","timestep","covariate","value"</li><li>list other.par, other parameters to `abundance model`, such as alpha.matrix, lambda.cov.matrix, alpha.cov.matrix.</li></ul> Note that the fields "species", "site", and "covariate" should be unique identifiers, character or numeric.
`timesteps`	number of timesteps to project
`abundance.model`	a function that accepts parameters from sp.par, a set of initial abundances, and optionally other parameters. The function returns the projected abundances at t+1
`return.seeds`	boolean flag, whether the prediction should return number of seeds (i.e. $N_{i,t+1}$, eq. 1 of Lanuza et al. 2018), or number of adult individuals, (i.e. $N_{i,t+1} * g$)

Value

dataframe with fields "timestep","site","sp","abundance", giving the expected abundance for each species, timestep, and site.

`salinity` *Salinity measurements*

Description

A dataset containing the integrated salinity over the season for each plant species.

Usage

```
data(salinity)
```

Format

A data frame with 324 rows and 5 variables

Details

- plot: plot number
- subplot: subplot code
- year: year of measurement
- precip: anual precipitation
- sum_salinity: summatory of the salinity values integrated througth the species lifespan.

Note

For details, see Lanuza et al. 2018 Ecology Letters.

`SpeciesFitness` *Fitness of a species*

Description

Calculates the fitness of a species following eq. S2 of Godoy et al. (2014). If germ.rate and survival.rate are provided, it calculates nu according to eq. 4 of Godoy et al. (2014), for annual plants. Otherwise, if only lambda is provided, nu = lambda

Usage

```
SpeciesFitness(lambda, germ.rate = NULL, survival.rate = NULL,
  competitive.response)
```

Arguments

`lambda`	per capita fecundity of the species in the absence of competition
`germ.rate`	optional, germination rate of the species
`survival.rate`	optional, annual survival of ungerminated seed in the soil
`competitive.response`	parameter reflecting the species' sensitivity to competition

Value

single numeric value, species fitness

`species_rates` *Species germination and survival rates*

Description

A dataset containing germination and survival rates for each plant species. It also includes a column with the scientific names, and their associated codes.

Usage

```
data(species_rates)
```

Format

A data frame with 20 rows and 4 variables

Details

- species: binomial name
- code: four-letter code used in other datasets
- germination: germination rate
- seed.survival: annual survival of ungerminated seed in the soil

Note

For details, see Lanuza et al. 2018 Ecology Letters.

Index