Package 'aae.pop'

January 31, 2026

Type Package

Title Flexible Population Dynamics Simulations

Version 0.2.0

Date 2026-01-27

Maintainer Jian Yen <jdl.yen@gmail.com>

Description Simulate population dynamics from realistically complex matrix population models in a plug-and-play fashion. Supports aspatial and spatially implicit models with one or more species and time-varying covariates, stochasticity, density dependence, additions or removals of individuals, interspecific interactions, and metapopulations.

License Apache License 2.0

URL https://aae-stats.github.io/aae.pop/, https://github.com/aae-stats/aae.pop

BugReports https://github.com/aae-stats/aae.pop/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, abind, cubature, future.apply, mc2d, nleqslv, rlang

Suggests knitr, DiagrammeR, testthat, covr, rmarkdown, remotes, scales

VignetteBuilder knitr

RoxygenNote 7.3.2

Language en-GB

NeedsCompilation no

Author Jian Yen [aut, cre, cph],
Arthur Rylah Institute for Environmental Research [fnd]

Repository CRAN

Date/Publication 2026-01-31 18:40:08 UTC

Contents

`add_remove` *Specify additions or removals in models of population dynamics*

Description

Specify additions or removals from the population vector that occur before (`add_remove_pre`) or after the update step (`add_remove_post`).

Usage

```
add_remove_pre(masks, funs)

add_remove_post(masks, funs)
```

Arguments

`masks`	a logical matrix or vector (or list of these) defining cells affected by `funs`. See Details and `masks`
`funs`	a function or list of functions with one element for each element of `masks`. See Details

Details

add_remove_pre specifies a function that operates on the population vector prior to the population update step. Examples might include fatalities (recorded in absolute numbers), removals, or additions to the population that occur prior to the update (shifting from one generation to the next).

add_remove_post is the same as add_remove_pre but operates on the population vector after the population update step.

masks are logical vectors with one element for each class. Additional details on masks are provided in masks.

funs takes only one argument, the population abundances n prior (add_remove_pre) or following (add_remove_post) all other updates in a given iteration/generation. This allows direct additions or removals to the population vector, potentially based on external arguments (e.g., mass mortality events or harvesting).

Additional arguments to functions are supported and can be passed to simulate with the args argument.

Value

add_remove_pre or add_remove_post object specifying an additions/removals process for use with dynamics

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# remove up to 10 individuals from stages 4 and 5 prior to the
#   matrix update
removals <- add_remove_pre(
  masks = all_classes(popmat, dims = 4:5),
  funs = \(x) ifelse(x > 10, x - 10, 0)
)

# update the dynamics object
dyn <- update(dyn, removals)

# simulate trajectories
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# and plot
plot(sims)

# remove up to 10 individuals from stages 4 and 5 after to the
#   matrix update
```

```
removals <- add_remove_post(
  masks = all_classes(popmat, dims = 4:5),
  funs = \(x) ifelse(x > 10, x - 10, 0)
)

# update the dynamics object (can't update because that will
#   include the add_remove_pre as well)
dyn <- dynamics(popmat, removals)

# simulate trajectories
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# and plot
plot(sims)
```

`covariates` *Specify covariate dependence in models of population dynamics*

Description

Specify relationship between a vector or matrix of covariates and vital rates.

Usage

```
covariates(masks, funs)

format_covariates(x, aux = NULL, names = NULL)
```

Arguments

`masks`	a logical matrix or vector (or list of these) defining cells affected by `funs`. See Details and `masks`
`funs`	a function or list of functions with one element for each element of `masks`. See Details
`x`	a vector, matrix, or data.frame of time-varying covariate values with one element or row per time step
`aux`	additional, static arguments to be passed to a covariates function
`names`	optional vector of names for each covariate included in `x`

Details

Masks must be of the same dimension as the population dynamics matrix and specify cells influenced by covariates according to `funs`. Additional details on masks are provided in `masks`.

Functions must take at least one argument, a vector or matrix representing the masked elements of the population dynamics matrix. Incorporating covariate values requires a second argument. Functions must return a vector or matrix with the same dimensions as the input, modified to reflect the effects of covariates on vital rates.

Additional arguments to functions are supported and can be passed to `simulate` with the `args`, `args.dyn`, or `args.fn` arguments.

`format_covariates` is a helper function that takes covariates and auxiliary values as inputs and returns a correctly formatted list that can be passed as `args` to `simulate`.

Value

`covariates` object specifying covariate effects on a matrix population model; for use with `dynamics`

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# can add covariates that influence vital rates
#   e.g., a logistic function
covars <- covariates(
  masks = transition(popmat),
  funs = function(mat, x) mat * (1 / (1 + exp(-10 * x)))
)

# simulate 50 random covariate values
xvals <- matrix(runif(50), ncol = 1)

# update the dynamics object and simulate from it.
#   Note that ntime is now captured in the 50 values
#   of xvals, assuming we pass xvals as an argument
#   to the covariates functions
dyn <- update(dyn, covars)
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  args = list(
    covariates = format_covariates(xvals)
  )
)

# and can plot these simulated trajectories
plot(sims)
```

density_dependence *Specify density dependence in models of population dynamics*

Description

Specify density dependence in vital rates (density_dependence) and in total abundances (density_dependence_n).

Usage

```
density_dependence(masks, funs, nmask = NULL)

density_dependence_n(masks, funs)
```

Arguments

masks	a logical matrix or vector (or list of these) defining cells affected by funs. See Details and masks
funs	a function or list of functions with one element for each element of masks. See Details
nmask	logical vector or list of vectors defining elements of the population vector affected by each mask-function pair. Intended primarily for internal use when scaling up processes in metapopulation

Details

density_dependence specifies standard density dependence on vital rates, such as scramble or contest competition or allee effects.

density_dependence_n is an alternative parameterisation of density dependence that acts directly on population abundances. Note that density_dependence_n has been superseded by add_remove_post.

Masks must be of the same dimension as the population dynamics matrix and specify cells influenced by density dependence according to funs. In the case of density_dependence_n, masks are logical vectors with one element for each class. Additional details on masks are provided in masks.

If using density_depenence, functions must take at least two arguments, a matrix x and a vector n, which represent the population dynamics matrix and the population abundances. Functions must return a matrix with the same dimensions as x, modified to reflect the effects of current abundances (n) on vital rates.

In the case of density_dependence_n, funs takes only one argument, the population abundances n following all other updates in a given iteration/generation. This allows rescaling of population abundances based on total abundance or through more complicated functions that depend on external arguments (e.g., mass mortality events or harvesting).

Additional arguments to functions are supported and can be passed to simulate with the args, args.dyn, or args.fn arguments.

Value

density_dependence object specifying covariate effects on a matrix population model; for use with dynamics

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# add some density dependence
dd <- density_dependence(
  masks = reproduction(popmat, dims = 4:5),
  funs = ricker(1000)
)

# update the dynamics object
dyn <- update(dyn, dd)

# simulate trajectories
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# and plot
plot(sims)
```

---

density_functions *Common forms of density dependence*

Description

Use pre-defined forms of density dependence based on common density-dependence functions.

Usage

```
beverton_holt(k, exclude = NULL)

ricker(k, exclude = NULL)
```

Arguments

k	carrying capacity used to define models of density dependence. See details for currently implemented models and their parameters.
exclude	vector of classes to exclude from calculation of total population density. Defaults to NULL, in which case all classes are used

Details

Additional functions are provided to define common forms of density dependence. Currently implemented models are the Ricker model and Beverton-Holt model, both with a single parameter k.

Value

functions that can be used with density_dependence to specify common models of density dependence

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# add some density dependence
dd <- density_dependence(
  masks = reproduction(popmat, dims = 4:5),
  funs = ricker(1000)
)

# update the dynamics object
dyn <- update(dyn, dd)

# simulate trajectories
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# and plot
plot(sims)
```

---

dispersal *Specify dispersal between populations in a metapopulation model*

Description

Specify dispersal between populations, including stochasticity and density dependence in dispersal parameters

Usage

```
dispersal(
  kernel,
  stochasticity_masks = NULL,
```

```
  stochasticity_funs = NULL,
  density_masks = NULL,
  density_funs = NULL,
  proportion = FALSE
)
```

Arguments

kernel	numeric matrix specifying the probability of specific classes moving between two populations. Matrices have the same columns-move-to-rows structure as in the population dynamics matrices described in dynamics, so a non-zero value in cell (a, b) denotes a transition from class b in the source population to class a in the receiving population
stochasticity_masks	a logical matrix or list of logical matrices defining cells affected by stochasticity_funs. See Details and masks
stochasticity_funs	a function or list of functions with one element for each element of stochasticity_masks. See Details
density_masks	a logical matrix or list of logical matrices defining cells affected by density_funs. See Details and masks
density_funs	a function or list of functions with one element for each element of density_masks. See Details
proportion	logical indicating whether kernel is specified in absolute probabilities or as a proportion of the source population (defaults to FALSE). If TRUE, values in kernel are calculated as a proportion of the total probability an individual exits that class at any given time step

Value

dispersal object specifying probabilities of movement between populations in a metapopulation matrix model; for use with metapopulation

Examples

```
# define some populations, all with identical vital rates
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- lapply(
  1:3,
  function(i) dynamics(popmat)
)

# define metapopulation structure with populations
#   1 and 3 dispersing into population 2
```

```
pop_structure <- matrix(0, nrow = 3, ncol = 3)
pop_structure[1, 2] <- 1
pop_structure[3, 2] <- 1

# define dispersal between populations
dispersal_matrix <- matrix(0, nrow = nclass, ncol = nclass)
dispersal_matrix[survival(dispersal_matrix, dims = 20:25)] <- 0.2
pop_dispersal1 <- dispersal(dispersal_matrix, proportion = TRUE)
pop_dispersal2 <- dispersal(dispersal_matrix, proportion = FALSE)
pop_dispersal <- list(pop_dispersal1, pop_dispersal2)

# create metapopulation object
metapop <- metapopulation(pop_structure, dyn, pop_dispersal)

# simulate without covariates
sims <- simulate(metapop, nsim = 10)

# and plot the simulated trajectories
plot(sims)
```

`dynamics` *Create and update population dynamics objects*

Description

Define population dynamics from a matrix and additional objects that determine covariate effects, density dependence, and forms of stochasticity.

Usage

```
dynamics(matrix, ...)

## S3 method for class 'dynamics'
update(object, ...)

is.dynamics(x)
```

Arguments

`matrix`	a matrix of vital rates specifying transitions between ages or stages. Specified in the format ntp1 = A A is the matrix and nt is the vector of abundances, so that values in a given column and row denote a transition from that column to that row
`...`	additional objects used to define population dynamics. Must be one or more of `covariates`, `replicated_covariates`, `environmental_stochasticity`, `demographic_stochasticity`, `density_dependence`, `add_remove_pre`, or `add_remove_post`. Note that `density_dependence_n` is equivalent to `add_remove_post`.
`object`	a `dynamics` object
`x`	an object to pass to `is.dynamics`

Details

A call to dynamics defines an object of class dynamics, which can be used to simulate population trajectories with the simulate function. The plot function is supported and will generate a general life-cycle diagram based on the defined population dynamics.

A compiled dynamics object can be updated to change any of the included processes with the update function.

Value

dynamics object containing a matrix population model and all associated processes

Examples

```
# define a population
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# and plot this
if (rlang::is_installed("DiagrammeR")) {
  plot(dyn)
}
```

---

emps *Calculate expected minimum population size (EMPS) for a simulate object*

Description

Calculate expected minimum population size (EMPS) for a simulate object

Usage

```
emps(sims, subset = NULL, times = NULL, fun = mean, ...)
```

Arguments

sims	an object returned from simulate
subset	integer vector denoting the population classes to include in calculation of population abundance. Defaults to all classes
times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations
fun	function used to calculate average over all replicate trajectories. Defaults to mean. Alternatives might include median or min
...	additional arguments passed to fun

Details

Expected minimum population size (EMPS) is the average minimum value of all replicate trajectories. This value represents an expected lower bound on population sizes over all generations, accounting for variation among replicates. Abundances can be specified for all population classes or for a subset of classes.

Value

a single value representing the expected minimum population size for a simulation

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn, nsim = 1000)

# calculate expected minimum population size
emps(sims)

# calculate expected minimum population size for 4 and 5 year
#   olds only
emps(sims, subset = 4:5)

# calculate expected minimum population size but ignore first 10 years
emps(sims, times = 11:51)

# calculate expected minimum population size based on median
emps(sims, fun = median)
```

---

exps *Calculate expected population size for a* `simulate` *object based on generic functions (ExPS)*

Description

Calculate expected population size for a `simulate` object based on generic functions (ExPS)

Usage

```
exps(
  sims,
  subset = NULL,
  times = NULL,
  fun_within = mean,
  fun_among = mean,
  ...
)
```

Arguments

sims	an object returned from simulate
subset	integer vector denoting the population classes to include in calculation of population abundance. Defaults to all classes
times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations
fun_within	function used to summarise a single trajectory. Must return a single value. Defaults to mean
fun_among	function used to summarise over all replicate trajectories. Defaults to mean. Alternatives might include median or min
...	additional arguments passed to fun_within and fun_among. If these conflict, a wrapper function could be used to define expected arguments for each function

Details

Expected population size (ExPS) is a highly flexible generalisation of emps and represents a two-level summary that first summarises individual population trajectories and then summarises these values over all replicates. Abundances can be specified for all population classes or for a subset of classes.

Value

a single value representing the expected statistic applied to the population sizes generated with simulate

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
```

```
sims <- simulate(dyn, nsim = 1000)

# calculate expected population size
exps(sims)

# calculate expected population size for 4 and 5 year
#   olds only
exps(sims, subset = 4:5)

# calculate expected population size but ignore first 10 years
exps(sims, times = 11:51)

# calculate expected population size based on median
exps(sims, fun_among = median)

# calculate expected maximum population size based on median
exps(sims, fun_within = max, fun_among = median)

# calculate exps with conflicting quantile functions, handling
#   conflicting arguments with wrapper functions
quant1 <- function(x, p1, ...) {
  quantile(x, prob = p1)
}
quant2 <- function(x, p2, ...) {
  quantile(x, prob = p2)
}
exps(
  sims,
  fun_within = quant1, fun_among = quant2, p1 = 0.25, p2 = 0.75
)
```

`get_cdf` *Calculate the cumulative distribution function of a summary statistic across all iterations of a* `simulate` *object*

Description

Calculate the cumulative distribution function of a summary statistic across all iterations of a `simulate` object

Usage

```
get_cdf(sims, subset = NULL, times = NULL, n = 100, fn = min, ...)
```

Arguments

`sims`	an object returned from `simulate`
`subset`	`integer` vector denoting the population classes to include in calculation of population abundance. Defaults to all classes

times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations
n	integer specifying number of threshold values to use in default case when threshold is not specified. Defaults to 100
fn	function to apply to each iteration. Defaults to min
...	additional arguments passed to fn

Details

get_cdf is a faster and more general alternative to the risk_curve function. get_cdf can be used to calculate the cumulative distribution of any summary statistic. For example, the cumulative distribution of the minimum population size is equivalent to a risk curve. Summary statistics for get_cdf are extracted from a simulate object and represent the cumulative distribution of that statistic over all replicate trajectories at any time step within a set period. Abundances can be specified for all population classes or for a subset of classes.

Value

a data.frame containing a prob column that indicates the probability the population will fall below the threshold value in the value column

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn, nsim = 1000)

# calculate distribution of minimum population sizes (default)
get_cdf(sims)

# calculate distribution of maximum population sizes
get_cdf(sims, fn = max)

# calculate distribution of the 90th percentile of
#   population sizes
get_cdf(sims, fn = quantile, prob = 0.9)

# calculate distribution of minimum population sizes
#   but ignore first 10 years
get_cdf(sims, fn = max, times = 11:51)
```

get_pdf *Calculate the probability density of a summary statistic across all iterations of a* simulate *object*

Description

Calculate the probability density of a summary statistic across all iterations of a simulate object

Usage

```
get_pdf(sims, subset = NULL, times = NULL, n = 100, fn = min, ...)
```

Arguments

sims	an object returned from simulate
subset	integer vector denoting the population classes to include in calculation of population abundance. Defaults to all classes
times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations
n	integer specifying number of threshold values to use in default case when threshold is not specified. Defaults to 100
fn	function to apply to each iteration. Defaults to min
...	additional arguments passed to fn

Details

get_pdf and get_cdf are faster and more general alternatives to the risk_curve function. get_pdf can be used to calculate the probability distribution of any summary statistic. For example, the probability distribution of the minimum population size is the density-based equivalent of a risk curve (the function get_cdf can be used to get the true equivalent). Summary statistics for get_pdf are extracted from a simulate object and represent the distribution of that statistic over all replicate trajectories at any time step within a set period. Abundances can be specified for all population classes or for a subset of classes.

Value

a data.frame containing a prob column that indicates the probability density that abundances will be in the vicinity of the threshold value in the value column

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)
```

```
# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn, nsim = 1000)

# calculate distribution of minimum population sizes (default)
get_pdf(sims)

# calculate distribution of maximum population sizes
get_pdf(sims, fn = max)

# calculate distribution of the 90th percentile of
#   population sizes
get_pdf(sims, fn = quantile, prob = 0.9)

# calculate distribution of minimum population sizes
#   but ignore first 10 years
get_pdf(sims, fn = max, times = 11:51)
```

---

masks *Isolate elements of population dynamics models*

Description

Helper functions to isolate particular components of a population dynamics model, such as the reproduction terms, transition/growth terms, or particular life stages from an abundance vector, such as pre- or post-reproductive stages.

Usage

```
reproduction(matrix, dims = NULL)

survival(matrix, dims = NULL)

transition(matrix, dims = NULL)

all_cells(matrix, dims = NULL)

all_classes(matrix, dims = NULL)

combine(...)
```

Arguments

matrix	a population dynamics matrix for which a particular mask is required. Only used to determine mask dimensions, so can be any matrix with appropriate dimensions

dims	a numeric value or vector identifying subsets of cells to include in a given mask
...	a set of masks or masking functions to be combined into a single mask by one of the combine methods

Value

mask object used to define the cells affected by a process included in dynamics

Examples

```
# define a population
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# pull out reproductive elements
reproduction(popmat)

# what if only 4 and 5 year olds reproduce?
reproduction(popmat, dims = 4:5)

# define survival elements
survival(popmat)

# what if 1 and 2 year olds always transition?
survival(popmat, dims = 3:5)

# and transitions
transition(popmat)

# combine transitions and reproduction of 4 and 5 year olds
combine(reproduction(popmat, dims = 4:5), transition(popmat))

# can also mask the population vector in this way
# pull out all classes
all_classes(popmat)

# and just 3-5 year olds
all_classes(popmat, dims = 3:5)
```

---

metapopulation *Create a metapopulation dynamics object*

Description

Define population dynamics for multiple populations of a single species linked by dispersal (a metapopulation).

Usage

```
metapopulation(structure, dynamics, dispersal)

is.metapopulation(x)
```

Arguments

structure	binary or logical matrix denoting dispersal links between populations. Columns move to rows, so a 1 or TRUE in cell (a, b) denotes movement from population b to population a
dynamics	a dynamics object or list of dynamics objects with one element for each population (each column/row of structure). If a single dynamics object is provided, it is recycled over all required populations
dispersal	object created with dispersal. dispersal objects describe movements between populations and can include class-specific movements and density-dependent movements. dispersal objects must be a list with one element for each link in structure. These links are interpreted in column-major order, so that dispersal objects must be ordered by links in column 1, then column 2, and so on
x	an object to pass to is.metapopulation

Details

The metapopulation function connects multiple populations through known dispersal probabilities, handling standardisations of dispersal probabilities (if required) and updating masks and functions for all processes defined within each population. Further details on the definition of dispersal terms are provided in dispersal.

Covariates can be included in metapopulation models. The default behaviour is for all populations to share a single set of covariates, with covariate associations and masks defined separately for each population. A workaround to the assumption of shared covariates is included in the examples, below. Including covariates on dispersal probabilities requires covariate associations and masks defined on the combined metapopulation model. This approach is possible but currently untested.

Value

metapopulation object containing a matrix metapopulation model; for use with simulate

Examples

```
# define some populations, all with identical vital rates
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- lapply(
  1:3,
  function(i) dynamics(popmat)
```

```
)

# define metapopulation structure with populations
#   1 and 3 dispersing into population 2
pop_structure <- matrix(0, nrow = 3, ncol = 3)
pop_structure[1, 2] <- 1
pop_structure[3, 2] <- 1

# define dispersal between populations
dispersal_matrix <- matrix(0, nrow = nclass, ncol = nclass)
dispersal_matrix[survival(dispersal_matrix, dims = 20:25)] <- 0.2
pop_dispersal1 <- dispersal(dispersal_matrix, proportion = TRUE)
pop_dispersal2 <- dispersal(dispersal_matrix, proportion = FALSE)
pop_dispersal <- list(pop_dispersal1, pop_dispersal2)

# create metapopulation object
metapop <- metapopulation(pop_structure, dyn, pop_dispersal)

# simulate without covariates
sims <- simulate(metapop, nsim = 2)

# simulate with shared covariates
# define a covariates function
covar_fn <- function(mat, x) {
  mat * (1 / (1 + exp(-0.5 * (x + 10))))
}

# and some covariates
xsim <- matrix(rnorm(20), ncol = 1)

# update the population dynamics objects with covariates
dyn <- lapply(
  dyn,
  update,
  covariates(masks = transition(dyn[[1]]$matrix), funs = covar_fn)
)

# (re)create metapopulation object
metapop <- metapopulation(pop_structure, dyn, pop_dispersal)
sims <- simulate(
  metapop,
  nsim = 2,
  args = list(covariates = format_covariates(xsim))
)

# simulate with separate covariates
#  (requires re-definition of covariate functions)
new_fn <- function(i) {
  force(i)
  function(mat, x) {
    mat * (1 / (1 + exp(-0.5 * (x[i] + 10))))
  }
}
```

```
new_fn <- lapply(
  1:3,
  new_fn
)

# update the population dynamics objects with covariates
dyn <- lapply(
  dyn,
  update,
  covariates(masks = transition(dyn[[1]]$matrix), funs = covar_fn)
)

# (re)create metapopulation object
metapop <- metapopulation(pop_structure, dyn, pop_dispersal)

# and simulate with one column of predictors for each population
xsim <- matrix(rnorm(60), ncol = 3)
sims <- simulate(
  metapop,
  nsim = 2,
  args = list(covariates = format_covariates(xsim))
)
```

`multispecies` *Create a population dynamics object with multiple species*

Description

Define population dynamics for multiple species from a set of single-species `dynamics` objects and defined pairwise interactions.

Usage

```
multispecies(...)

is.multispecies(x)

is.interaction(x)
```

Arguments

`...`	`pairwise_interaction` objects defining a set of pairwise interactions between species
`x`	an object to pass to `is.multispecies`

Value

`multispecies` object containing a multispecies matrix population model; for use with `simulate`

Examples

```
# define population matrices for three species
sp1_mat <- rbind(
  c(0, 0, 2, 4, 7), # reproduction from 3-5 year olds
  c(0.25, 0, 0, 0, 0), # survival from age 1 to 2
  c(0, 0.45, 0, 0, 0), # survival from age 2 to 3
  c(0, 0, 0.70, 0, 0), # survival from age 3 to 4
  c(0, 0, 0, 0.85, 0) # survival from age 4 to 5
)
sp2_mat <- rbind(
  c(0, 0, 4), # reproduction from 3 year olds
  c(0.25, 0, 0), # survival from age 1 to 2
  c(0, 0.45, 0) # survival from age 2 to 3
)
sp3_mat <- rbind(
  c(0, 0, 2, 4, 7, 10), # reproduction from 3-6 year olds
  c(0.25, 0, 0, 0, 0, 0), # survival from age 1 to 2
  c(0, 0.45, 0, 0, 0, 0), # survival from age 2 to 3
  c(0, 0, 0.70, 0, 0, 0), # survival from age 3 to 4
  c(0, 0, 0, 0.85, 0, 0), # survival from age 4 to 5
  c(0, 0, 0, 0, 0.75, 0) # survival from age 5 to 6
)

# define population dynamics objects for each species
sp1_dyn <- dynamics(sp1_mat)
sp2_dyn <- dynamics(sp2_mat)
sp3_dyn <- dynamics(sp3_mat)

# define multispecies interactions as masks/functions
# - species 1 influencing transition probabilities of species 3
mask_1v3 <- transition(sp3_mat)

# basic Beverton-Holt function
fun_1v3 <- function(x, n) {
  # n is the population vector of the source population (sp 1)
  x / (1 + x * sum(n[3:5]) / 100) # focus on adults
}

# - species 3 influencing reproduction of species 2
mask_3v2 <- reproduction(sp2_mat, dims = 3)

# basic Ricker function
fun_3v2 <- function(x, n) {
  # n is the population vector of the source population (sp 3)
  x * exp(1 - sum(n[1:2]) / 50) / exp(1) # focus on juveniles
}

# combine masks and functions into pairwise_interaction objects
sp_int1v3 <- pairwise_interaction(sp3_dyn, sp1_dyn, mask_1v3, fun_1v3)
sp_int3v2 <- pairwise_interaction(sp2_dyn, sp3_dyn, mask_3v2, fun_3v2)

# compile a multispecies dynamics object
```

```
multisp_dyn <- multispecies(sp_int1v3, sp_int3v2)

# simulate
sims <- simulate(multisp_dyn, nsim = 100)

# and can plot these simulated trajectories for each species
plot(sims, which = 1)
```

`pairwise_interaction` *Specify interactions between two species*

Description

Define population dynamics for multiple species from a set of single-species `dynamics` objects and defined pairwise interactions.

Usage

```
pairwise_interaction(target, source, masks, funs)
```

Arguments

`target`	population whose vital rates are affected by the pairwise interaction
`source`	population whose abundances affect the vital rates of `target`
`masks`	masks defining which vital rates are influenced by each function
`funs`	functions that take vital rates and abundances of the `source` population as inputs and return scaled vital rates

Value

`pairwise_interaction` object specifying links between species; for use with `multispecies`

`pr_extinct` *Calculate (quasi-)extinction risk for a* `simulate` *object*

Description

Calculate (quasi-)extinction risk for a `simulate` object

Usage

```
pr_extinct(sims, threshold = 0, subset = NULL, times = NULL)
```

Arguments

sims	an object returned from simulate
threshold	integer or numeric denoting the threshold population size below which a population is considered functionally extinct. Defaults to 0
subset	integer vector denoting the population classes to include in calculation of population abundance. Defaults to all classes
times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations

Details

Quasi-extinction risk is the probability of decline below some specified abundance threshold. This probability is extracted from a simulate object as the proportion of replicate trajectories that fall below this threshold at any time step within a set period. Abundances can be specified for all population classes or for a subset of classes.

Value

a single numeric value representing the probability a population will decline below the threshold size

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn, nsim = 1000)

# calculate quasi-extinction risk at a threshold population size
#   of 100 individuals
pr_extinct(sims, threshold = 100)

# repeat previous but focused on 4 and 5 year olds only
pr_extinct(sims, threshold = 100, subset = 4:5)

# repeat previous but ignore first 10 years
pr_extinct(sims, threshold = 100, times = 11:51)
```

`replicated_covariates` *Specify replicate-specific covariate dependence in models of population dynamics*

Description

Specify relationship between a vector or matrix of covariates and vital rates, but with a different covariate value for each replicate (i.e., each value of `nsim` in `simulate`)

Usage

```
replicated_covariates(masks, funs)
```

Arguments

`masks`	a logical matrix or vector (or list of these) defining cells affected by `funs`. See Details and `masks`
`funs`	a function or list of functions with one element for each element of `masks`. See Details

Details

Masks must be of the same dimension as the population dynamics matrix and specify cells influenced by covariates according to `funs`. Additional details on masks are provided in `masks`.

Functions must take at least one argument, a vector or matrix representing the masked elements of the population dynamics matrix. Incorporating covariate values requires a second argument. Functions must return a vector or matrix with the same dimensions as the input, modified to reflect the effects of covariates on vital rates.

Additional arguments to functions are supported and can be passed to `simulate` with the `args` argument.

`format_covariates` is a helper function that takes covariates and auxiliary values as inputs and returns a correctly formatted list that can be passed as `args` to `simulate`.

`replicated_covariates` operates identically to `covariates` except that it allows for a different value of the covariate applied to each replicate trajectory. This specification can incorporate complex structures, such as temporal dynamics in environmental stochasticity and correlated uncertainty in vital rates.

Value

`replicated_covariates` object specifying replicate-specific covariate effects on a matrix population model; for use with `dynamics`

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# can add covariates that influence vital rates
#   e.g., a logistic function
covars <- replicated_covariates(
  masks = transition(popmat),
  funs = \(mat, x) mat * (1 / (1 + exp(-10 * x)))
)

# simulate 50 random covariate values for each replicate (each
#   value of nsim, set to 100 below)
xvals <- matrix(runif(50 * 100), ncol = 100)

# update the dynamics object and simulate from it.
#   Note that ntime is now captured in the 50 values
#   of xvals, assuming we pass xvals as an argument
#   to the covariates functions
dyn <- update(dyn, covars)
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  args = list(
    replicated_covariates = format_covariates(xvals)
  )
)

# and can plot these simulated trajectories
plot(sims)
```

`risk_curve` *Calculate (quasi-)extinction risk at multiple thresholds for a* `simulate` *object*

Description

Calculate (quasi-)extinction risk at multiple thresholds for a `simulate` object

Usage

```
risk_curve(sims, threshold = NULL, subset = NULL, times = NULL, n = 100)
```

Arguments

sims	an object returned from simulate
threshold	integer or numeric vector denoting the set of threshold population sizes used to define the risk curve. Defaults to n evenly spaced values from 0 to the maximum observed abundance
subset	integer vector denoting the population classes to include in calculation of population abundance. Defaults to all classes
times	integer vector specifying generations to include in calculation of extinction risk. Defaults to all simulated generations
n	integer specifying number of threshold values to use in default case when threshold is not specified. Defaults to 100

Details

Risk curves represent pr_extinct at multiple threshold population sizes simultaneously. This gives an expression of risk of population declines below a range of values. Risk curves are extracted from a simulate object as the proportion of replicate trajectories that fall below each threshold value at any time step within a set period. Abundances can be specified for all population classes or for a subset of classes.

The get_cdf function is a much faster way to generate risk curves for almost all use cases. The exception is when the threshold argument is used to specify threshold values that are not evenly spaced.

Value

a named vector containing the threshold values (names) and the probability the population will fall below these threshold values

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn, nsim = 100)

# calculate risk curve
risk_curve(sims, n = 10)
```

rng *Random number generators not available in existing R packages*

Description

Draw random numbers from unusual distributions, such as on the unit or non-negative real line with known means and standard deviations.

Usage

```
rmultiunit(
  n,
  mean,
  sd,
  Sigma = NULL,
  Omega = NULL,
  perfect_correlation = FALSE
)

rmultiunit_from_real(
  n,
  mean_real,
  sd_real = NULL,
  Sigma_chol = NULL,
  perfect_correlation = FALSE
)

runit_from_real(n, mean_real, sd_real)

runit(n, mean, sd)

unit_to_real(unit_mean, unit_sd)
```

Arguments

n	number of random draws to simulate. Each draw is a vector of values with length equal to `length(mean)` and `length(sd)` and the resulting output has `n` rows and `length(mean)` columns
mean	vector of mean values on the unit scale
sd	vector of positive standard deviations
Sigma	optional covariance matrix with dimensions of `length(mean)` by `length(mean)` defining covariances between each pair of values in `mean`. Note that only the correlation structure is retained from `Sigma`, so that standard deviations are still required
Omega	optional correlation matrix with dimensions of `length(mean)` by `length(mean)` defining correlations between each pair of values in `mean`

perfect_correlation
: logical, if TRUE and Sigma and Omega are NULL, then all values in each replicate (row) are perfectly correlated with known mean and standard deviation. If FALSE, then all values in each replicate are completely uncorrelated

mean_real
: vector of mean values converted to real-line equivalents

sd_real
: vector of standard deviations converted to real-line equivalents

Sigma_chol
: Cholesky decomposition of covariance matrix converted to real-line equivalent

unit_mean
: vector of mean values on the unit interval

unit_sd
: vector of standard deviations on the unit interval

Details

The r*unit family of functions support simulation of values on the unit interval based on a known mean, sd, and correlation structure. runit and runit_from_real are vectorised univariate functions, and rmultiunit and rmultiunit_from_real are multivariate versions of these same functions. runit and rmultiunit provide simulated values on the unit line with specified means, standard deviations, and correlation/covariance structure (in the case of rmultiunit).

The *_from_real versions of these functions are helpers that use pre-transformed estimates of parameters on the real line, calculated with unit_to_real. These functions are exported because unit_to_real, called within runit and rmultiunit, is slow. Separating this into a separate step allows less frequent calculations of this transformation using function or dynamic versions of args in simulate.

unit_to_real converts means and standard deviations from their values on the unit line to their equivalent values on the real line.

The use of the different versions of these functions is illustrated in the Macquarie perch example on the package [website](https://aae-stats.github.io/aae.pop/).

Value

a vector or matrix of random draws from the r*unit set of functions

Examples

```
# rmultiunit generates multivariate draws constrained to
#   the unit interval, with known mean, standard deviation,
#   and (optionally) covariance/correlation structure
rmultiunit(n = 10, mean = c(0.25, 0.5, 0.75), sd = c(0.1, 0.4, 0.25))

# add in a correlation structure
omega_set <- cbind(
  c(1, 0.25, 0.01),
  c(0.25, 1, 0.5),
  c(0.01, 0.5, 1)
)
rmultiunit(
  n = 10,
  mean = c(0.25, 0.5, 0.75),
  sd = c(0.1, 0.4, 0.25),
```

```
  Omega = omega_set
)
```

`simulate` *Simulate single or multispecies population dynamics in R*

Description

Simulate population dynamics for one or more species defined by `dynamics` objects.

Usage

```
## S3 method for class 'dynamics'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  ...,
  init = NULL,
  options = list(),
  args = list(),
  .future = FALSE
)

is.simulation(x)

is.simulation_list(x)
```

Arguments

`object`	a `dynamics` object created with `dynamics` or from a subsequent call to `multispecies` or `metapopulation`. Alternatively, `object` can be the output of a call to `simulate` in the case of `summary`
`nsim`	the number of replicate simulations (default = 1)
`seed`	optional seed used prior to initialisation and simulation to give reproducible results
`...`	ignored; included for consistency with `simulate` generic method
`init`	an array of initial conditions with one row per replicate and one column per population stage. If `obj` has been created with `multispecies`, initial conditions can be provided as a list or array with one element or slice per species, or as a matrix, in which case all species are assumed to share the same initial conditions. Defaults to `NULL`, in which case initial conditions are generated randomly according to `options()$aae.pop_initialisation`

options	a named list of simulation options. Currently accepted values are: - ntime the number of time steps to simulate, ignored if obj includes a covariates (default = 50) - keep_slices logical defining whether to keep intermediate population abundances or (if FALSE) to return only the final time slice - tidy_abundances a function to handle predicted abundance data that may be non-integer. Defaults to identity; suggested alternatives are floor, round, or ceiling - initialise_args a list of arguments passed to the function used to initialise abundance trajectories. Only used if init = NULL. Defaults to options()$aae.pop_lambda, which specifies lambda for Poisson random draws. The default initialisation function is defined by options()$aae.pop_initialisation. - update a function to update abundances from one time step to the next. Defaults to options()$aae.pop_update.
args	named list of lists passing arguments to processes defined in object, including interaction for multispecies objects. Lists (up to one per process) can contain a mix of static, dynamic, and function arguments. Dynamic arguments must be lists with one element per time step. Function arguments must be functions that calculate arguments dynamically in each generation based on from the population dynamics object, population abundances, and time step in each generation. All other classes (e.g., single values, matrices, data frames) are treated as static arguments. Covariates contained in numeric vectors, matrices, or data frames can be formatted as dynamic arguments with the format_covariates function. args for multispecies objects must have one element per species (defaults will expand automatically if not provided)
.future	flag to determine whether the future package should be used to manage updates for multispecies models (an embarrassingly parallel problem)
x	an object to pass to is.simulation or is.simulation.list

Value

simulation object containing replicate simulations from a matrix population model. plot and subset methods are defined for simulation objects

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# can extract standard population matrix summary stats
lambda <- Re(eigen(popmat)$values[1])

# define a dynamics object
dyn <- dynamics(popmat)
```

```
# simulate from this (50 time steps, 100 replicates)
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# plot the simulated trajectories
plot(sims)

# add some density dependence
dd <- density_dependence(
  masks = reproduction(popmat, dims = 4:5),
  funs = ricker(1000)
)

# update the dynamics object
dyn <- update(dyn, dd)

# simulate again
sims <- simulate(dyn, nsim = 100, options = list(ntime = 50))

# and plot
plot(sims)

# what if we want to add initial conditions?
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  options = list(ntime = 50),
)

# and plot again
plot(sims)

# note that there is only one trajectory now because
#   this simulation is deterministic.
#
# let's change that by adding some environmental stochasticity
envstoch <- environmental_stochasticity(
  masks = list(
    reproduction(popmat, dims = 4:5),
    transition(popmat)
  ),
  funs = list(
    \(x) rpois(n = length(x), lambda = x),
    \(x) runif(n = length(x), min = 0.9 * x, max = pmin(1.1 * x, 1))
  )
)

# update the dynamics object and simulate from it
dyn <- update(dyn, envstoch)
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
```

```
  nsim = 100,
  options = list(ntime = 50),
)

# can also add covariates that influence vital rates
#   e.g., a logistic function
covars <- covariates(
  masks = transition(popmat),
  funs = \(mat, x) mat * (1 / (1 + exp(-10 * x)))
)

# simulate 50 random covariate values
xvals <- matrix(runif(50), ncol = 1)

# update the dynamics object and simulate from it.
#   Note that ntime is now captured in the 50 values
#   of xvals, assuming we pass xvals as an argument
#   to the covariates functions
dyn <- update(dyn, covars)
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  args = list(covariates = format_covariates(xvals))
)

# a simple way to add demographic stochasticity is to change
#   the "updater" that converts the population at time t
#   to its value at time t + 1. The default in aae.pop
#   uses matrix multiplication of the vital rates matrix
#   and current population. A simple tweak is to update
#   with binomial draws. Note that this also requires a
#   change to the "tidy_abundances" option so that population
#   abundances are always integer values.
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  options = list(
    update = update_binomial_leslie,
    tidy_abundances = floor
  ),
  args = list(covariates = format_covariates(xvals))
)

# and can plot these again
plot(sims)
```

stochasticity *Specify environmental and demographic stochasticity in models of population dynamics*

Description

Specify environmental stochasticity (random variation in vital rates) and demographic stochasticity (random variation in population outcomes).

Usage

```
environmental_stochasticity(masks, funs)

demographic_stochasticity(masks, funs)
```

Arguments

masks	a logical matrix or vector (or list of these) defining cells affected by funs. See Details and masks
funs	a function or list of functions with one element for each element of masks. See Details

Details

Masks must be of the same dimension as the population dynamics matrix (in the case of environmental stochasticity) or have one element for each class (in the case of demographic stochasticity). Masks specify cells influenced by stochasticity according to funs. Additional details on masks are provided in masks.

Functions must have at least one argument, a population dynamics matrix for environmental stochasticity or a vector of population abundances for demographic stochasticity. Functions must return an output of the same dimensions as the input, modified to reflect the effects of stochasticity on vital rates or population abundances.

Additional arguments to functions are supported and can be passed to simulate with the args, args.dyn, or args.fn arguments.

Value

environmental_stochasticity or demographic_stochasticity object specifying the way in which stochasticity should be included in a matrix population model; for use with dynamics

Examples

```
# define a population matrix (columns move to rows)
nclass <- 5
popmat <- matrix(0, nrow = nclass, ncol = nclass)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
```

```
dyn <- dynamics(popmat)

# note that there is only one trajectory now because
#   this simulation is deterministic.
#
# let's change that by adding some environmental stochasticity
envstoch <- environmental_stochasticity(
  masks = list(
    reproduction(popmat, dims = 4:5),
    transition(popmat)
  ),
  funs = list(
    \(x) rpois(n = length(x), lambda = x),
    \(x) runif(n = length(x), min = 0.9 * x, max = pmin(1.1 * x, 1))
  )
)

# update the dynamics object and simulate from it
dyn <- update(dyn, envstoch)
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  options = list(ntime = 50),
)

# a simple way to add demographic stochasticity is to change
#   the "updater" that converts the population at time t
#   to its value at time t + 1. The default in aae.pop
#   uses matrix multiplication of the vital rates matrix
#   and current population. A simple tweak is to update
#   with binomial draws. Note that this also requires a
#   change to the "tidy_abundances" option so that population
#   abundances are always integer values.
sims <- simulate(
  dyn,
  init = c(50, 20, 10, 10, 5),
  nsim = 100,
  options = list(
    update = update_binomial_leslie,
    tidy_abundances = floor
  )
)
```

`updaters` *Functions for a single time-step update*

Description

Define how population abundances are updated from one time step to the next. Functions can take any form but will only be vectorised across replicates in limited situations.

Usage

```
update_crossprod(pop, mat)

update_binomial_leslie(pop, mat)

update_multinomial(pop, mat)
```

Arguments

pop	current state of the population
mat	matrix of vital rates used to update population state

Details

Updaters can be changed through the options argument to simulate and can also be changed globally for an R session by changing the global option with, e.g., options(aae.pop_update = update_binomial_leslie)

update_crossprod updates abundances with a direct matrix multiplication that does not include any form of demographic stochasticity. This is the fastest update option and will vectorise across replicates if the population matrix is not expanded by environmental_stochasticity or density_dependence.

update_binomial_leslie updates abundances with a direct RNG draw that combines update with demographic stochasticity, assuming a Leslie matrix.

update_multinomial updates abundances with a direct RNG draw that combines update with demographic stochasticity, allowing for general matrix forms (slower than update_binomial_leslie).

Value

a matrix containing population abundances in each stage of a matrix population model. Contains one row for each replicate population trajectory and one column for each population stage

Examples

```
# define a basic population
nstage <- 5
popmat <- matrix(0, nrow = nstage, ncol = nstage)
popmat[reproduction(popmat, dims = 4:5)] <- c(10, 20)
popmat[transition(popmat)] <- c(0.25, 0.3, 0.5, 0.65)

# define a dynamics object
dyn <- dynamics(popmat)

# simulate with the default updater
sims <- simulate(dyn)

# simulate with a multinomial updater
sims <- simulate(dyn, options = list(update = update_multinomial))
```

Index