Package ‘steps’

October 14, 2022

Type Package

Title Spatially- and Temporally-Explicit Population Simulator

Version 1.3.0

Date 2022-10-03

Maintainer Casey Visintin <casey.visintin@unimelb.edu.au>

Description Software to simulate population change across space and time. Visintin et al. (2020) <doi:10.1111/2041-210X.13354>.

BugReports https://github.com/steps-dev/steps/issues

URL https://github.com/steps-dev/steps

Depends R (>= 3.4.0)

License GPL (>= 2)

Imports Rcpp, raster, future, future.apply, rasterVis, viridisLite, memuse

LinkingTo Rcpp

RoxygenNote 7.2.1

Suggests testthat, fields, knitr, rmarkdown, foreach

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

LazyData true

NeedsCompilation yes

Author Casey Visintin [aut, cre] (<https://orcid.org/0000-0003-2245-8998>),
Nick Golding [ctb],
Skipton Woolley [ctb]

Repository CRAN

Date/Publication 2022-10-04 23:30:02 UTC

R topics documented:

`ceiling_density` *Ceiling-based density dependence*

Description

In-built density dependence function that constrains the number of individuals in a cell based on the carrying capacity of that cell in a timestep. Note, carrying_capacity must be provided in the landscape object to use this function (see landscape). Only specified stages that contribute to density dependence are considered in the calculations and excess individuals are removed from only the contributing stages. This type of density dependence only affects the population once it reaches the carrying capacity. While population size is below carrying capacity, the population grows according to the transition matrix.

Usage

```
ceiling_density(stages = NULL)
```

Arguments

`stages`	which life-stages contribute to density dependence and are removed in a timestep - default is all

Examples

```
# Cap the population at carrying capacity with only the second and third
# life stage used in calculations to determine density dependence.

## Not run:
cap_population <- ceiling_density(stages = c(2, 3))

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat), density_dependence = cap_population)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`cellular_automata_dispersal` *Cellular automata dispersal*

Description

The cellular_automata_dispersal function simulates movements of individuals using rule-based cell movements. In each cell that has population, every individual up to a specified proportion of the total population attempts to move. For each step from a specified minimum up to a specified maximum number of movements, a weighted draw of four directions, based on habitat suitability, is made and then the destination cell is checked for available carrying capacity. If there is carrying capacity available, the individual moves to the cell, if not, it remains in its current cell. This is repeated until the maximum number of cell movements is reached. If no cell is found with available carrying capacity, the individual remains in the source cell.

Usage

```
cellular_automata_dispersal(
  max_cells = Inf,
  min_cells = max_cells,
  dispersal_proportion = set_proportion_dispersing(),
  barriers = NULL,
  use_suitability = TRUE,
  carrying_capacity = "carrying_capacity"
)
```

Arguments

`max_cells` the maximum number of cell movements that each individual in each life stage can disperse in whole integers.

`min_cells` the minimum number of cell movements that each individual in each life stage will disperse in whole integers.

`dispersal_proportion` a built-in or custom function defining the proportions of individuals that can disperse in each life stage.

`barriers` the name of a spatial layer in the landscape object that contains cell values between 0 (no barrier) and 1 (full barrier) Any values between 0 and 1 indicate the permeability of the barrier.

`use_suitability` should habitat suitability be used to control the likelihood of individuals dispersing into cells? The default is TRUE. Note, if a barrier map is also provided, the suitability map is multiplied with the barrier map to generate a permeability map of the landscape.

`carrying_capacity` the name of a spatial layer in the landscape object that specifies the carrying capacity in each cell.

Details

This function allows the use of barriers in the landscape to influence dispersal. The function is computationally efficient, however, because as individuals are dispersed, performance scales with the population sizes in each cell across a landscape and the maximum number of cell movements.

The maximum number of cell movements in cellular automata dispersal does not correspond exactly to the distance decay of a dispersal kernel, since cellular automata dispersal depends on the permeability of the landscape, and is interrupted on reaching a cell with available capacity (above the minimum specified number of cell movements). A heuristic that can be used to determine a reasonable number of steps from a mean dispersal distance ‘d‘ and cell resolution ‘res‘ is: ‘max_cells = round(2 * (d / (res * 1.25)) ^ 2)‘. This corresponds approximately to the number of cell-steps in an infinite, homogenous landscape with no early stopping, for which d is the mean end-to-end dispersal distance of all individuals.

Rather than relying on this value, we recommend that the user experiment with the `max_cells` and `min_cells` parameters to find a value such that the the mean dispersal distance in a reasonably realistic simulation corresponds with field estimates of mean dispersal distances.

Examples

```
# Example of cellular automata dispersal where the 2nd and 3rd life stages
# disperse up to a maximum of 100 cells but dispersal is affected by
# barriers (in this case roads). The road rasters have values of 0 for
# large roads (no dispersal across barrier) and 0.5 for smaller roads
# (reduced dispersal across barrier).

## Not run:
ca_dispersal <- cellular_automata_dispersal(max_cells = c(0, 100, 100), barriers = "roads")

ls <- landscape(population = egk_pop,
                suitability = egk_hab,
                carrying_capacity = egk_k,
                "roads" = egk_road)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = ca_dispersal,
                          density_dependence = ceiling_density())

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

---

compare_emp *Compare minimum expected populations*

Description

Compare minimum expected populations from two or more ’simulation_results’ objects.

Usage

```
compare_emp(
  x,
  ...,
```

```
  show_interval = TRUE,
  interval = 95,
  all_points = FALSE,
  simulation_names = NULL
)
```

Arguments

x	a simulation_results object
...	additional simulation results objects
show_interval	should the interval bars be shown on the plot? Default is TRUE.
interval	the desired confidence interval representing the uncertainty around the expected minimum population estimates from simulation comparisons; expressed as a whole integer between 0 and 100 (default value is 95).
all_points	should the expected minimum populations from all simulation replicates be shown on the plot? Default is FALSE.
simulation_names	an optional character vector of simulation names to override the defaults

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

# Create populations dynamics with and without ceiling density dependence
pd1 <- population_dynamics(change = growth(egk_mat),
                           dispersal = kernel_dispersal(max_distance = 1000,
                  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 500)),
                           density_dependence = ceiling_density())
pd2 <- population_dynamics(change = growth(egk_mat),
                           dispersal = kernel_dispersal(max_distance = 3000,
                  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1500)))

# Run first simulation with ceiling density dependence and three replicates
sim1 <- simulation(landscape = ls,
                   population_dynamics = pd1,
                   habitat_dynamics = NULL,
                   timesteps = 20,
                   replicates = 3)

# Run second simulation without ceiling density dependence and three replicates
sim2 <- simulation(landscape = ls,
                   population_dynamics = pd2,
                   habitat_dynamics = NULL,
                   timesteps = 20,
                   replicates = 3)

compare_emp(sim1, sim2)

## End(Not run)
```

competition_density *Competition density function*

Description

Adjusts the life-stage transition matrix in each cell based on the carrying capacity in the cell and a density dependence function - default is Beverton-Holt. The user may specify which life-stages are affected by density dependence. If R_max is not provided this is calculated from the local cell-based transition matrices internally. By providing initial stable age distribution values, performance can be increased as the function internally calculates these values through optimisation.

Usage

```
competition_density(
  stages = NULL,
  mask = NULL,
  R_max = NULL,
  stable_age = NULL
)
```

Arguments

stages	which life-stages contribute to density dependence - default is all
mask	a matrix of boolean values (TRUE/FALSE), equal in dimensions to the life-stage transition matrix and specifying which vital rates (i.e. survival and fecundity) are to be modified by the function
R_max	optional value of maximum growth rate (lambda) if known
stable_age	optional vector of stable age distributions if known

Examples

```
# Vital rates (survival and fecundity) modified based on approach to carrying capacity
# by the 2nd and 3rd life stages.

## Not run:
mod_fun <- competition_density(stages = c(2, 3))

ls <- landscape(population = egk_pop, suitability = NULL, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat, transition_function = mod_fun))

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

density_dependence_dispersing
Density-dependent proportions of populations dispersing

Description

The proportion of populations dispersing will be density dependent in a simulation. Proportions of populations in each life stage dispersing is adjusted based on available carrying capacity. If life-stages are set by the population_density_dependence_functions, these will be used to determine how close the population is to carrying capacity. If no life-stages are set or density dependence is set to NULL in population_dynamics, the function will consider all life-stages in the calculation.

Usage

```
density_dependence_dispersing(maximum_proportions = 1)
```

Arguments

`maximum_proportions`
: A single value or vector of the maximum proportions (between zero and one) of individuals in each life stage that disperse - default is 1. If maximum proportions are specified as a single number, then all life-stages use that value, however, a vector of maximum proportions (equal in length to the number of life-stages) can also be specified. Maximum proportions are multiplied by the calculated proportions based on carrying capacity so to prevent stages from dispersing, set corresponding values to zero.

Value

An object of class `dispersal_proportion_function`

Examples

```
# Example of a proportion function that disperses all populations based on their approach
# to carrying capacity

## Not run:
prop_dispersal <- density_dependence_dispersing()

kb_dispersal <- kernel_dispersal(dispersal_proportion = prop_dispersal,
                      max_distance = 2000,
                  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1000))

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat), dispersal = kb_dispersal)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)
```

```
## End(Not run)
```

`dispersal_kernel` *Create a dispersal function*

Description

A dispersal kernel function is a mathematical representation of how species redistribute across the landscape.

A common dispersal kernel is provided in the software for the user to select, however, a user may also provide a custom written dispersal kernel.

See Also

- exponential_dispersal_kernel) for a (negative) exponential dispersal kernel

---

`dispersal_proportion_function` *Create a proportion dispersing function*

Description

A proportion dispersing function generates the proportions of species that disperse from cells based on landscape features.

Details

The default `set_proportion_dispersing` function and parameters returns full dispersal for all life stages. Additional proportion dispersing functions are provided in the software for the user to select, however, a user may also provide a custom written proportion dispersing function. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

See Also

- set_proportion_dispersing controls the proportions of each life-stage that disperse
- density_dependence_dispersing proportions of dispersing populations are controlled by approach to carrying capacity

disturbance *Disturbance*

Description

Modifies the landscape by multiplying habitat suitability values by a sum of previous disturbances. Since disturbances can act in a single timestep, or have lasting effects, the user can specify an 'effect time' of disturbances.

Usage

```
disturbance(disturbance_layers, effect_time = 1)
```

Arguments

`disturbance_layers`
: the name of spatial layer(s) in the landscape object with disturbances used to alter the habitat object for each timestep (number of layers must match the intended timesteps)

`effect_time`
: the number of timesteps that the disturbance layer will act on the habitat object (e.g. '3' will combine the effects of previous two timesteps to increase the overall effect) - the default is 1.

Examples

```
# Road building (stored in the landscape object and called "roads") acts on the landscape
# each year.

## Not run:
road_effect <- disturbance(disturbance_layers = "roads", effect_time = 1)

ls <- landscape(population = egk_pop, suitability = egk_hab, "roads" = egk_road)

pd <- population_dynamics(change = growth(egk_mat))

sim <- simulation(landscape = ls,
           population_dynamics = pd,
           habitat_dynamics = list(road_effect),
           timesteps = 20)

plot(sim, object = "suitability", type = "raster", timesteps = 1:9)

## End(Not run)
```

egk — *Eastern Grey Kangaroo example data*

Description

Example data for simulating spatial population dynamics of Eastern Grey Kangaroos in a hypothetical landscape.

Usage

```
egk_hab

egk_pop

egk_k

egk_mat

egk_mat_stoch

egk_sf

egk_fire

egk_origins

egk_destinations

egk_road
```

Format

Misc data

An object of class `RasterStack` of dimension 35 x 36 x 3.

An object of class `RasterLayer` of dimension 35 x 36 x 1.

An object of class `matrix` (inherits from `array`) with 3 rows and 3 columns.

An object of class `matrix` (inherits from `array`) with 3 rows and 3 columns.

An object of class `RasterStack` of dimension 35 x 36 x 120.

An object of class `RasterBrick` of dimension 35 x 36 x 20.

An object of class `RasterLayer` of dimension 35 x 36 x 1.

An object of class `RasterLayer` of dimension 35 x 36 x 1.

An object of class `RasterBrick` of dimension 35 x 36 x 20.

Details

- **egk_hab** A raster layer containing the predicted relative habitat suitability for the Eastern Grey Kangaroo.
- **egk_pop** A raster stack containing initial populations for each life-stage of the Eastern Grey Kangaroo.
- **egk_k** A raster layer containing the total number of Eastern Grey Kangaroos each grid cell can support.
- **egk_mat** A matrix containing the survival and fecundity of Eastern Grey Kangaroos at each of three life-stages - juvenile, subadult, and adult.
- **egk_mat_stoch** A matrix containing the uncertainty around survival and fecundity of Eastern Grey Kangaroos at each of three life-stages - juvenile, subadult, and adult.
- **egk_sf** A raster stack containing values for modifying survival and fecundities - each is raster is named according to the timestep and position of the life-stage matrix to be modified.
- **egk_fire** A raster stack containing values for modifying the habitat - in this case the proportion of landscape remaining after fire.
- **egk_origins** A raster stack containing locations and counts of where to move individual kangaroos from.
- **egk_destinations** A raster stack containing locations and counts of where to move individual kangaroos to.
- **egk_road** A raster stack containing values for modifying the habitat - in this case the proportion of habitat remaining after the construction of a road.

---

exponential_dispersal_kernel

Negative exponential dispersal kernel

Description

This function determines the proportion of redistribution based on distance.

Usage

```
exponential_dispersal_kernel(distance_decay = 0.5, normalize = FALSE)
```

Arguments

distance_decay	(exponential dispersal parameter) controls the rate at which the population disperses with distance
normalize	(exponential dispersal parameter) should the normalising constant be used - default is FALSE.

Value

An object of class `dispersal_function`

Examples

```
## Not run:
dists <- seq(0, 100, 1)

exp_dispersal_fun <- exponential_dispersal_kernel(distance_decay = 50)

plot(dists, exp_dispersal_fun(dists), type = 'l')

## End(Not run)
```

---

extract_spatial — *Extract spatial object from a 'simulation_results' object*

Description

The simulation results object is a list of lists containing spatial (and other) objects and is organised by the following tree diagram:

- Replicate
  - Timestep
    - Population Raster Stack
      - Life-Stage Raster
    - Habitat Suitability Raster (or Stack)
      - Habitat Raster (if stack is used)
    - Carrying Capacity Raster
    - Other Raster Stack
      - Raster
    - ...

Usage

```
extract_spatial(
  x,
  replicate = 1,
  timestep = 1,
  landscape_object = "population",
  stage = 1,
  misc = 1
)
```

Arguments

`x`	a simulation_results object
`replicate`	which replicate to extract from a `simulation_results` object
`timestep`	which timestep to extract from a `simulation_results`

landscape_object
: which landscape object to extract from a simulation_results object - can be specified by name (e.g. "suitability") or index number

stage
: which life-stage to extract from a simulation_results object - only used for 'population' components of the landscape object

misc
: which misc object to extract from a simulation_results object - only used for additional spatial objects added to a landscape (e.g. disturbance layers)

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Extract the population raster for the second life-stage in the first
# replicate and ninth timestep
extract_spatial(sim, replicate = 1, timestep = 9, stage = 2)

## End(Not run)
```

---

fast_dispersal *Fast diffusion-based dispersal*

Description

The fast_dispersal function uses kernel-based dispersal to modify the population with a user-defined diffusion distribution and a fast-fourier transformation (FFT) computational algorithm. It is computationally efficient and very fast, however, only useful for situations where dispersal barriers or arrival based on habitat or carrying capacity are not required (e.g. a homogeneous landscape or where diffusion alone is sufficient to explain dispersal patterns). Dispersal is not constrained to suitable habitat or available carrying capacity.

Usage

```
fast_dispersal(
  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 0.1),
  dispersal_proportion = set_proportion_dispersing()
)
```

Arguments

`dispersal_kernel`
: a single built-in or user-defined distance dispersal kernel function.

`dispersal_proportion`
: a built-in or custom function defining the proportions of individuals that can disperse in each life stage.

Examples

```
# Example of fast kernel-based dispersal where all life stages disperse.
# The default dispersal kernel uses a decay parameter to control how far
# populations disperse. Note proportions of populations to disperse are
# controlled by approach to carrying capacity.

## Not run:
fft_dispersal <- fast_dispersal(dispersal_proportion = density_dependence_dispersing(),
                 dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1000))

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = fft_dispersal,
                          density_dependence = ceiling_density())

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`fire_effects` *Fire effects with regeneration*

Description

Modifies the landscape by multiplying habitat suitability values by a weighted sum of previous fire intensities based on a user specified regeneration function. By default, the regenerative function is an inverse linear relationship to time, however, this function can be replaced with a response that takes into account other factors of habitat restoration (e.g. growth/re-growth curves of vegetation).

Usage

```
fire_effects(
  fire_layers,
  effect_time = 3,
  regeneration_function = function(time) {
     -time
 }
)
```

Arguments

`fire_layers` the name(s) of spatial layer(s) in the landscape object with fire disturbances used to alter the habitat object for each timestep (number of layers must match the intended timesteps)

`effect_time` the number of timesteps that the fire layer will act on the habitat object

`regeneration_function`
a function that determines how fast the landscape will regenerate after a fire event

Examples

```
# Fire (stored in the landscape object and called "fires") acts on the landscape for
#five years with an exponentially decaying intensity.

## Not run:
regen <- function (time) {-exp(time)}

plot(1:5, regen(1:5), type = "l")

fire <- fire_effects(fire_layers = "fires", effect_time = 5, regeneration_function = regen)

ls <- landscape(population = egk_pop, suitability = egk_hab, "fires" = egk_fire)

pd <- population_dynamics(change = growth(egk_mat))

sim <- simulation(landscape = ls,
           population_dynamics = pd,
           habitat_dynamics = list(fire),
           timesteps = 20)

plot(sim, object = "suitability", type = "raster", timesteps = 1:9)

## End(Not run)
```

`growth` *Population growth*

Description

This function applies negative or positive growth to the population using matrix multiplication. Stochasticity can be added to cell-based transition matrices or globally. Users can also specify a built-in or custom function to modify the transition matrices throughout a simulation. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

Usage

```
growth(
  transition_matrix,
  global_stochasticity = 0,
  local_stochasticity = 0,
  transition_function = NULL,
  transition_order = c("fecundity", "survival"),
  two_sex = FALSE
)
```

Arguments

`transition_matrix`
: A symmetrical age-based (Leslie) or stage-based (Lefkovitch) population structure matrix.

`global_stochasticity, local_stochasticity`
: Either scalar values or matrices (with the same dimension as `transition_matrix`) specifying the variability in the transition matrix either for populations in all grid cells (`global_stochasticity`) or for each grid cell population separately (`local_stochasticity`). Values supplied here are the standard deviation of a truncated normal distribution where the mean is the value supplied in the transition matrix.

`transition_function`
: A function to specify or modify life-stage transitions at each timestep. See transition_function.

`transition_order`
: Order of transitions performed in growth function. This behaviour is only applied when demographic stochasticity is set to "full" (default) and transitions are applied sequentially. By default "fecundity" is performed first (calculating the number of new individuals to be added to the populations), then "survival" is applied. The final population is the sum of these. Users should be cautious of specifying "survival" to be performed first as typically survival of reproductive stages will already be accounted for in the fecundity values of the transition matrix.

`two_sex`
: Does the transition matrix include life stages for two sexes (i.e. male and female)? Default is FALSE which assumes a single sex matrix (e.g. females only).

Examples

```
# Example of a growth function that changes the populations based on a transition matrix that
# is subject to global stochasticity.

## Not run:
stoch_growth <- growth(transition_matrix = egk_mat, global_stochasticity = egk_mat_stoch)

ls <- landscape(population = egk_pop, suitability = NULL, carrying_capacity = NULL)
```

```
pd <- population_dynamics(change = stoch_growth)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

---

habitat_dynamics_functions
Functions to modify the habitat in a landscape object.

Description

Pre-defined functions to operate on habitat suitability (and carrying capacity if a function is used) during a simulation.

See Also

- disturbance to modify the suitability of a landscape with user provided spatially-explicit layers
- fire_effects

---

kernel_dispersal *Kernel-based dispersal*

Description

The kernel_dispersal function employs a probabilistic kernel-based dispersal algorithm to modify the population using a user-defined diffusion distribution (see dispersal_kernel), arrival probability layers (e.g. habitat suitability), and growth limiting layers (e.g. carrying capacity). This function is much slower than the fast_dispersal, however, respects dispersal limitations which may be more ecologically appropriate. Further, the kernel-based dispersal function utilises a mechanism to optimise computational performance in which it switches between pre-allocating cell movements based on the available memory of the host computer (faster but more memory intensive) or executing cell movements in sequence (slower but less memory intensive).

Usage

```
kernel_dispersal(
  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1),
  max_distance = NULL,
  arrival_probability = c("both", "suitability", "carrying_capacity", "none"),
  dispersal_proportion = set_proportion_dispersing()
)
```

Arguments

`dispersal_kernel`
: a single built-in or user-defined distance dispersal kernel function.

`max_distance`
: the maximum distance that each life stage can disperse in spatial units of the landscape (in kernel-based dispersal this truncates the dispersal curve). Setting a reasonable number will increase the performance of a simulation by reducing the number of cells that need to be calculated in distance matrices.

`arrival_probability`
: the name of a spatial layer in the landscape object that controls where individuals can disperse to (e.g. "suitability") or "none" to allow individuals to disperse to all non-NA cells. The default is to use both the habitat suitability and carrying capacity layers. When this option is selected, the arrival probability in each cell is calculated by multiplying the habitat suitability by one minus the proportion of space taken up in the cell (total population of life stages contributing to density dependence divided by the carrying capacity).

`dispersal_proportion`
: a built-in or custom function defining the proportions of individuals that can disperse in each life stage.

Examples

```
# Example of kernel-based dispersal where only the 3rd life stage
# disperses up to a maximum distance of 2000 meters. Dispersal is affected
# by both habitat suitability and carrying capacity (default). The default
# dispersal kernel uses a decay parameter to control how far populations disperse.

## Not run:
kb_dispersal <- kernel_dispersal(max_distance = 2000,
                  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1000))

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kb_dispersal,
                          density_dependence = ceiling_density())

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`landscape` *Create a landscape object.*

Description

A landscape object is used to store spatially-explicit information on population, habitat suitability, carrying_capacity and miscellaneous landscape information.

Usage

```
landscape(population, suitability = NULL, carrying_capacity = NULL, ...)
```

Arguments

- `population` a raster stack (grid cell-based) with one layer for each life stage.
- `suitability` an optional raster layer or stack (multiple layers) containing habitat suitability values for all cells in a landscape. Note, using a raster stack assumes that the user has provided a layer for each intended timestep in a simulation.
- `carrying_capacity` an optional raster layer specifying carrying capacity values for all cells in a landscape or a function defining how carrying capacity is determined by habitat suitability.
- `...` named raster objects representing different aspects of the landscape used to modify the landscape object in a simulation. Note, this is intended to store objects that are accessed by dynamic functions and used to modify the landscape in a simulation. Also, further arguments passed to or from other methods.

Details

A landscape object is modified in each timestep of a simulation. During a simulation, population, habitat suitability or carrying capacity in a landscape object are changed based on dynamic functions selected or created by the user.

Value

An object of class `landscape`

Examples

```
# Example of setting up a landscape object.

## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat))

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

modified_transition *Spatially-explicit transition function*

Description

In the built-in `modified_transition` function, the values of fecundity and survival in local cell-based transition matrices are multiplied by values in the named spatial objects for each cell. The spatial objects can be rasters that are stored in the landscape object.

Usage

```
modified_transition(survival_layer = NULL, fecundity_layer = NULL)
```

Arguments

`survival_layer` the name of a spatial layer in the landscape object used to modify survival values (i.e. non-zero values in rows other than the first).

`fecundity_layer` the name of a spatial layer in the landscape object used to modify fecundity values (i.e. non-zero values in the first row).

Details

The behaviour of the function is to modify any non-zero values in the first row by the "fecundity_layer" and non-zero values in rows other than the first by the "survival_layer". This is irrespective of the type of matrix or any assumptions made by the user in creating the transition matrix. For example, if the transition matrix values include both the probabilities of surviving AND growing into the next stage, these can NOT be modified individually. This operation would require the use of a custom function - see the "Creating custom *steps* functions" vignette for more information.

Note, this function will not work if two-sex transition matrices are specified in a simulation. This function can be modified, however, to accommodate two-sex models - review the population_change function and see the "Creating custom *steps* functions" vignette for more information.

Value

An object of class `transition_function`

Examples

```
# Vital rates (survival and fecundity) modified based on habitat suitability.

## Not run:
mod_fun <- modified_transition(survival_layer = "suitability", fecundity_layer = "suitability")

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = NULL)

pd <- population_dynamics(change = growth(egk_mat, transition_function = mod_fun))
```

```
simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`mortality` *Directly affect populations*

Description

This function modifies a population by a mortality spatial layer included in a steps landscape object. The mortality layer consists of values from 0???1 and modifies the population by multiplying the population of a cell by the value of the corresponding cell in a mortality layer. For example, a cell with ten individuals before the mortality function is applied, and corresponding mortality layer cell with a value of 0.2, would have two individuals remaining after modification. Note, rounding also occurs after modification using a ceiling method (i.e the largest whole integer is retained).

Usage

```
mortality(mortality_layer, stages = NULL)
```

Arguments

`mortality_layer`
: the name of spatial layer(s) in the landscape object with mortality proportions used to alter the populations for each timestep. If a stack of rasters is used then the number of layers must match the intended number of timesteps in the simulation.

`stages`
: which life-stages are modified - default is all

Examples

```
# Modify populations in all life-stages with fire intensity.

## Not run:
fire_mortal <- mortality(mortality_layer = "fire", stages = NULL)

ls <- landscape(population = egk_pop,
                suitability = egk_hab,
                carrying_capacity = egk_k,
                "fire" = egk_fire)

pd <- population_dynamics(change = growth(egk_mat), modification = fire_mortal)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

plot.simulation_results

Plot the results of a simulation

Description

Methods to visually inspect the results of a simulation. Both linear graphs and spatial-explicit grids are generated for all timesteps to illustrate population changes through time and space. Note, this function can be wrapped in a *png()* call to write several images to disk for creating animations.

Usage

```
## S3 method for class 'simulation_results'
plot(x, replicates = 1, ...)
```

Arguments

`x`	a simulation_results object
`replicates`	which replicates to plot (default is one, or the first)
`...`	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                         dispersal_kernel = exponential_dispersal_kernel(
                                           distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the spatial distributions of total cell populations
plot(sim)

## End(Not run)
```

plot_hab_spatial *Plot habitat suitability spatial information*

Description

Plot spatial grids to illustrate habitat suitability changes through time.

Usage

```
plot_hab_spatial(x, replicate = 1, timesteps = NULL, ...)
```

Arguments

x	a simulation_results object.
replicate	replicate to plot - note, only one replicate can be plotted at a time. The default is to plot the first replicate
timesteps	timesteps to plot
...	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the population trajectories by life-stage
plot_hab_spatial(sim)

## End(Not run)
```

plot_k_spatial — *Plot carrying capacity spatial information*

Description

Plot spatial grids to illustrate carrying capacity changes through time.

Usage

```
plot_k_spatial(x, replicate = 1, timesteps = NULL, ...)
```

Arguments

x	a simulation_results object.
replicate	replicate to plot - note, only one replicate can be plotted at a time. The default is to plot the first replicate
timesteps	timesteps to plot
...	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the population trajectories by life-stage
plot_k_spatial(sim)

## End(Not run)
```

plot_k_trend *Plot carrying capacity (k) trend*

Description

Plot linear graphs to illustrate carrying capacity changes through time.

Usage

```
plot_k_trend(x, summary_stat = "mean", return_data = FALSE, ...)
```

Arguments

x	a simulation_results object
summary_stat	how to summarize the values across the landscape - "mean" (default) or "sum"
return_data	(TRUE/FALSE) should the data used to create the plots be returned?
...	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the carrying capacity trajectories
plot_k_trend(sim)

## End(Not run)
```

plot_pop_spatial — *Plot population spatial information*

Description

Plot spatial grids to illustrate population changes through time.

Usage

```
plot_pop_spatial(x, stage = 0, replicate = 1, timesteps = NULL, ...)
```

Arguments

`x`	a simulation_results object
`stage`	life-stage to plot - defaults to totals of all life stages. Set to zero for totals (i.e. sum of all life-stages).
`replicate`	replicate to plot - note, only one replicate can be plotted at a time. The default is to plot the first replicate
`timesteps`	timesteps to plot
`...`	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                          dispersal_kernel = exponential_dispersal_kernel(
                                            distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the population trajectories by life-stage
plot_pop_spatial(sim)

## End(Not run)
```

plot_pop_trend *Plot population trend*

Description

Plot linear graphs to illustrate population changes through time.

Usage

```
plot_pop_trend(x, stages = NULL, emp = FALSE, return_data = FALSE, ...)
```

Arguments

x	a simulation_results object
stages	life-stages to plot - by default all life-stages will be shown. Set to zero for totals (i.e. sums of all life-stages).
emp	(TRUE/FALSE) add a dashed line indicating the expected minimum population of the simulation (for multiple replicates only)
return_data	(TRUE/FALSE) should the data used to create the plots be returned?
...	further arguments passed to/from other methods

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

# Plot the population trajectories by life-stage
plot_pop_trend(sim)

# Plot the total population trajectory
plot_pop_trend(sim, stages = 0)

## End(Not run)
```

`population_change_functions` *How the population changes in a landscape.*

Description

Pre-defined or custom functions to define population change during a simulation. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

See Also

- growth is a default function for changing populations based on transition matrices and functions

`population_density_dependence_functions` *How the population responds to density dependence in a landscape.*

Description

Pre-defined or custom functions to define population density dependence (e.g. ceiling) during a simulation. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

See Also

- ceiling_density to cap populations at carrying capacities

`population_dispersal_functions` *How the population disperses in a landscape.*

Description

Pre-defined or custom functions to define population dispersal during a simulation. Each dispersal method uses different computing resources and may be applicable to different simulation scenarios. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

See Also

- kernel_dispersal for kernel-based diffusion dispersal using habitat suitability and/or carrying capacity to influence movements
- cellular_automata_dispersal for individual-based movements using rule-sets
- fast_dispersal for quick kernel-based diffusion dispersal without accounting for spatial heterogeneity

---

population_dynamics *Define population dynamics.*

Description

A `population_dynamics` object is used to describe how populations change in space and time.

Usage

```
population_dynamics(
  change = NULL,
  dispersal = NULL,
  modification = NULL,
  density_dependence = NULL,
 dynamics_order = c("change", "dispersal", "modification", "density_dependence")
)
```

Arguments

`change`	population_change_functions to define how population growth occurs at each timestep
`dispersal`	population_dispersal_functions to define how the population disperses at each timestep
`modification`	population_modification_functions to define any deterministic changes to the population - such as translocations or population control - at each timestep
`density_dependence`	population_density_dependence_functions to control density dependence effects on the population at each timestep
`dynamics_order`	the order in which the population dynamics should be executed on the landscape object - default is "change" -> "dispersal" -> "modification" -> "density_dependence". Note, if population dynamics are reordered, all dynamics must be listed in `dynamics_order`.

Details

A population_dynamics object is passed to simulation and defines how populations change between timesteps. Note, some dynamics functions can be executed at non-regular intervals (i.e. only timesteps explicitly defined by the user). The `population_dynamics` function is used to construct an object with several population dynamics functions and their associated parameters. These functions specify how the population in the landscape object will be modified throughout a simulation. The dynamics can be executed in any order that is specified by the user. It is cautioned that the order of dynamics will have implications depending on whether the user has assumed a post-breeding or pre-breeding census in the transition matrix. For more information on this, please refer to Kendall et al, (2019) *Ecological Applications*.

Value

An object of class `population_dynamics`

Examples

```
# Example of setting up population dynamics to only use a population change function.

## Not run:
ls <- landscape(population = egk_pop, suitability = NULL, carrying_capacity = NULL)

pd <- population_dynamics(change = growth(egk_mat))

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`population_modification_functions`

How the population is modified in a landscape.

Description

Pre-defined functions to define population modification (e.g. translocation) during a simulation.

See Also

- translocation for specifying explicit spatial and temporal movements of populations
- mortality for specifying explicit spatial and temporal changes to populations

set_proportion_dispersing

Set proportions of populations dispersing

Description

This function allows a user to specify what proportions of populations in each life-stage disperse. It operates similarly on all cells and in all timesteps throughout a simulation.

Usage

```
set_proportion_dispersing(proportions = 1)
```

Arguments

`proportions`	A single value or vector of proportions (between zero and one) of individuals in each life stage that disperse - default is 1. If proportions are specified as a single number, then all life-stages disperse with that proportion, however, a vector of proportions (equal in length to the number of life-stages) can also be specified. To prevent stages from dispersing, set corresponding values to zero.

Value

An object of class `dispersal_proportion_function`

Examples

```
# Example of a proportion function that disperses no population in the first life stage,
# 50% of the second, and 90% of the 3rd.

## Not run:
prop_dispersal <- set_proportion_dispersing(proportions = c(0, 0.5, 0.9))

kb_dispersal <- kernel_dispersal(dispersal_proportion = prop_dispersal,
                      max_distance = 2000,
                  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1000))

ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat), dispersal = kb_dispersal)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

simulation *Run a simulation*

Description

A simulation changes landscape objects based on selected dynamics over a specified number of timesteps.

Usage

```
simulation(
  landscape,
  population_dynamics,
  habitat_dynamics = list(),
  demo_stochasticity = c("full", "none"),
  timesteps = 3,
  replicates = 1,
  verbose = TRUE,
  future.globals = list()
)
```

Arguments

`landscape`	a landscape object representing the initial habitat and population
`population_dynamics`	a population_dynamics object describing how population changes over time
`habitat_dynamics`	optional list of functions to modify the landscape at each timestep - see habitat_dynamics_functions
`demo_stochasticity`	how should population rounding occur, if at all - "full" uses a multinomial draw to return rounded cell populations (default) whilst "none" returns non-integer cell populations (no rounding). Note, this parameter specification is used consistently throughout all functions in a simulation.
`timesteps`	number of timesteps used in one simulation
`replicates`	number of simulations to perform
`verbose`	print messages and progress to console? (default is TRUE)
`future.globals`	a list of custom functions, and objects called by the functions, that a user has created in the global environment for use in a simulation. Note this is only required when running simulations in parallel (e.g. plan(multisession)).

Value

An object of class `simulation_results`

Examples

```
## Not run:
ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

pd <- population_dynamics(change = growth(egk_mat),
                          dispersal = kernel_dispersal(max_distance = 2000,
                                        dispersal_kernel = exponential_dispersal_kernel(
                                          distance_decay = 1000)),
                          density_dependence = ceiling_density())

# Run a simulation with full demographic stochasticity and without any habitat
# dynamics for tewnty timesteps.
sim <- simulation(landscape = ls,
                  population_dynamics = pd,
                  habitat_dynamics = NULL,
                  timesteps = 20)

## End(Not run)
```

`steps` *Simulate population trajectories over space and time with dynamic functions.*

Description

Simulating shifts in species populations is an important part of ecological management. Species respond to spatial and temporal changes in the landscape resulting from environmental phenomena, managerial actions or anthropogenic activities. This data is crucial for modelling, however, current software that incorporates this information has limited flexibility, transparency, and availability. `steps` extends the features found in existing software and accepts common spatial inputs that are derived from many other existing software packages.

A simulation is run on a landscape using population dynamics functions contained in a population_dynamics object. habitat_dynamics_functions can also be added to the simulation to modify the habitat during a simulation.

`transition_function` *Create a growth transition function*

Description

A growth transition function defines how spatial objects or custom functions influence survival and fecundity. Two built-in functions are provided for the user to select, however, a user may also provide custom written functions to modify survival and fecundity throughout a simulation. Please see the tutorial vignette titled "Creating custom *steps* functions" for information on how to write custom functions for use in simulations.

See Also

- modified_transition to use rasters to modify survival and fecundity
- competition_density to use relationship to carrying capacity to modify survival and fecundity

---

`translocation` *Translocate populations*

Description

This function is used to move or introduce populations throughout a simulation. A user can specify which life-stages will be affected (default is all) and in which timesteps the translocations will take place. A warning will be generated if populations are not available where specified to translocate from.

Usage

```
translocation(
  origins_layer,
  destinations_layer,
  stages = NULL,
  effect_timesteps = 1
)
```

Arguments

`origins_layer`	the name of a spatial layer in the landscape object with the locations and number of individuals to translocate from. Note, this layer will have only zero values if individuals are being introduced from outside the study area
`destinations_layer`	the name of a spatial layer in the landscape object with the locations and number of individuals to translocate to. Note, this layer will have only zero values if individuals are being controlled (e.g. culling)
`stages`	which life-stages are modified - default is all
`effect_timesteps`	which timesteps in a single simulation do the translocations take place

Examples

```
# Modify populations in all life-stages using explicit layers of origin and destination populations
# in timesteps 5, 10, and 15.

## Not run:
trans_pop <- translocation(origins_layer = "origins",
                           destinations_layer = "destinations",
                           stages = NULL,
```

```
                              effect_timesteps = c(5, 10, 15))

ls <- landscape(population = egk_pop,
                suitability = NULL,
                carrying_capacity = NULL,
                "origins" = egk_origins,
                "destinations" = egk_destinations)

pd <- population_dynamics(change = growth(egk_mat), modification = trans_pop)

simulation(landscape = ls, population_dynamics = pd, habitat_dynamics = NULL, timesteps = 20)

## End(Not run)
```

`visualisation` *Visualise the results of a *steps* simulation*

Description

Visualising the results of a simulation is important to verify parameter assumptions and quantitative model behaviour. Both linear graphs indicating trends and spatial-explicit grids containing spatial arrangement of information can be generated to illustrate changes through time and space for populations, carrying capacity, and habitat suitability. The expected minimum populations (EMP) can also be compared for several different simulations.

Details

For plotting trends, see:

- `plot_pop_trend` to examine population changes
- `plot_k_trend` to examine carrying capacity changes

For plotting spatial information, see:

- `plot_pop_spatial` to examine population changes
- `plot_k_spatial` to examine carrying capacity changes
- `plot_hab_spatial` to examine habitat suitability changes

For plotting and comparing expected minimum populations, see:

- `compare_emp` to examine how different simulations compare

Index