Package 'steps'

April 2, 2019

Type Package

Title Spatially- and Temporally-Explicit Population Simulator

Version 0.2.1

Date 2019-04-02

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

Description Software to simulate population dynamics across space and time.

Depends R (>= 3.2.2)

License GPL (>= 2)

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

LinkingTo Rcpp

RoxygenNote 6.1.1

Suggests testthat, fields, knitr, rmarkdown, foreach

VignetteBuilder knitr

Encoding UTF-8

LazyData true

NeedsCompilation yes

Author Casey Visintin [aut, cre],
Nick Golding [ctb],
Skipton Woolley [ctb],
John Baumgartner [ctb]

Repository CRAN

Date/Publication 2019-04-02 11:30:03 UTC

R topics documented:

all_dispersing 2
egk 3
exponential_dispersal_kernel 4
habitat_dynamics_functions 5

landscape 6
modified_transition 7
population_change_functions 8
population_density_dependence_functions 9
population_dispersal_functions 9
population_dynamics 11
population_modification_functions 12
simulation 13
steps 15

Index **16**

`all_dispersing` *Create a proportion dispersing function*

Description

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

The default proportion dispersing function returns full dispersal (1.0) for all life stages. Additional proportion dispersing functions are also provided in the software for the user to select (see' carrying_capacity_dispersal), however, a user may also provide a custom written proportion dispersing function.

Usage

```
all_dispersing(proportions = 1)

carrying_capacity_dispersal()
```

Arguments

`proportions`	proportions of individuals in each life stage that disperse - default is 1

Value

An object of class `dispersal_proportion_function`

Examples

```
test_dispersal_function <- all_dispersing()

test_dispersal_function <- carrying_capacity_dispersal()
```

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_source

egk_sink

egk_road
```

Format

Misc data

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_source A raster stack containing locations and counts of where to move individual kangaroos from.

egk_sink 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` *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 (see exponential_dispersal_kernel), however, a user may also provide a custom written dispersal kernel.

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

```
test_dispersal_function <- exponential_dispersal_kernel()
```

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.

Usage

```
disturbance(disturbance_layers, effect_time = 1)

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

Arguments

disturbance_layers	the name(s) 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
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)
lag	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

```
library(steps)


# Use the disturbance function to modify the habitat using spatial
# layers (stored in the landscape object and called "logging"):

logging <- disturbance(disturbance_layers = "logging",
                   effect_time = 1)

# Use the fire_effects function to modify the habitat using spatial
# fire layers (stored in the landscape object and called "fires")
# and a regeneration function:
```

```
fire <- fire_effects(fire_layers = "fires",
                     lag = 5,
                     regeneration_function = function (time) {-time})
```

`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,
  ...)

is.landscape(x)
```

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.
`x`	a landscape object.

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

```
library(steps)
library(raster)

# Construct the landscape object
egk_ls <- landscape(population = egk_pop, suitability = egk_hab, carrying_capacity = egk_k)

# Test if object is of the type 'landscape'
is.landscape(egk_ls)
```

`modified_transition` *Create a growth transition function*

Description

A growth transition function defines how spatial objects or custom functions influence survival and fecundity. A user may select from built-in functions or provide a custom written function to modify survival and fecundity throughout a simulation.

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.

A commonly used `competition_density` dependence function is also provided in the software for the user to select, however, a user may also provide other custom written density dependence functions.

Usage

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

competition_density(transition_matrix, stages = NULL, mask = NULL,
  R_max = NULL, initial_stages = NULL)
```

Arguments

`transition_matrix`	a symmetrical age-based (Leslie) or stage-based population structure matrix.
`survival_layer`	the name of a spatial layer in the landscape object used to modify survival values.
`fecundity_layer`	the name of a spatial layer in the landscape object used to modify fecundity values.
`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
initial_stages	optional vector of stable age distributions if known

Value

An object of class transition_function

Examples

```
test_mod_transition <- modified_transition(egk_mat)

test_comp_transition <- competition_density(egk_mat)
```

---

population_change_functions

How the population changes in a landscape.

Description

Pre-defined or custom functions to define population change during a simulation.

Usage

```
growth(transition_matrix, global_stochasticity = 0,
  local_stochasticity = 0, transition_function = NULL)
```

Arguments

transition_matrix	A symmetrical age-based (Leslie) or stage-based population structure matrix.
global_stochasticity, local_stochasticity	either scalar values or matrices (with the same dimension as transition_matrix) specifying the variability (in standard deviations) in the transition matrix either for populations in all grid cells (global_stochasticity) or for each grid cell population separately (local_stochasticity)
transition_function	A custom function defined by the user specifying modifications to life-stage transitions at each timestep. See transition_function.

Examples

```
# Use a function to modify the
# population using life-stage transitions:

test_growth <- growth(egk_mat)
```

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.

Usage

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

Arguments

stages	which life-stages contribute to density dependence - default is all

Examples

```
test_pop_dd <- ceiling_density()
```

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.

Usage

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


 kernel_dispersal(dispersal_kernel = exponential_dispersal_kernel(distance_decay
  = 1), max_distance = Inf, arrival_probability = c("both",
  "suitability", "carrying_capacity", "none"),
  dispersal_proportion = all_dispersing())

cellular_automata_dispersal(max_distance = Inf,
  dispersal_kernel = exponential_dispersal_kernel(distance_decay = 1),
  dispersal_proportion = all_dispersing(), barriers_map = NULL,
```

```
  arrival_probability = "suitability",
  carrying_capacity = "carrying_capacity")
```

Arguments

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

`dispersal_proportion`
: proportions of individuals (0 to 1) that can disperse in each life stage. Can also be a custom function that relates the proportion dispersing to a spatial layer in the landscape object (e.g. carrying capacity).

`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) - must be specified.

`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.

`barriers_map`
: 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.

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

Details

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. In other words, organisms can disperse in all directions and to all cells in the landscape.

The kernel_dispersal function employs a probabilistic kernel-based dispersal algorithm to modify the population using a user-defined diffusion distribution, 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).

The cellular_automata_dispersal function modifies populations using rule-based cell movements. This function allows the use of barriers in the landscape to influence dispersal. The function is computationally efficient, however, because individuals are dispersed, performance scales with the population sizes in each cell across a landscape.

Examples

```
test_fast_dispersal <- fast_dispersal()

test_kern_dispersal <- kernel_dispersal()

test_ca_dispersal <- cellular_automata_dispersal()
```

---

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"))

is.population_dynamics(x)
```

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.
x	a population_dynamics object
...	further arguments passed to or from other methods

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.

Value

An object of class `population_dynamics`

Examples

```
test_pop_dynamics <- population_dynamics()

# Test if object is of the type 'population dynamics'
is.population_dynamics(test_pop_dynamics)
```

---

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.

Usage

```
translocation(source_layer, sink_layer, stages = NULL,
  effect_timesteps = 1)
```

Arguments

`source_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
`sink_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

```
# Use the translocation_population_dynamics object to modify the
# population using translocations:

test_translocation <- translocation(source_layer = "egk_source",
                                    sink_layer = "egk_sink",
                                    stages = NULL,
                                    effect_timesteps = 1)
```

`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", "deterministic_redistribution",
  "stochastic_redistribution", "none"), timesteps = 3, replicates = 1,
  verbose = TRUE)

is.simulation_results(x)

## S3 method for class 'simulation_results'
plot(x, object = "population",
  type = "graph", stages = NULL, animate = FALSE,
  timesteps = c(1:3), panels = c(3, 3), emp = FALSE, ...)

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

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); "deterministic_redistribution" redistributes the decimal overages to the cells with the largest overages; "stochastic_redistribution" redistributes the decimal overages to the cells based on binomial draws using the overages as probabilities; "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 or timesteps to display when plotting rasters

`replicates`
: number of simulations to perform

`verbose`
: print messages and progress to console? (default is TRUE)

`x`
: a simulation_results object

`object`
: the `simulation_results` object to plot - can be 'population' (default), 'suitability' or 'carrying_capacity'

`type`
: the plot type - 'graph' (default) or 'raster'

`stages`
: life-stages to plot - must be specified for 'raster' plot types; by default all life-stages will be plotted

`animate`
: if plotting type 'raster' would you like to animate the rasters?

`panels`
: the number of columns and rows to use when plotting raster timeseries - default is 3 x 3 (e.g. c(3,3))

`emp`
: should the expected minimum population of the simulation be plotted?

`...`
: further arguments passed to or from other methods

`replicate`
: which replicate to extract from a `simulation_results` object

`timestep`
: which timestep(s) to extract from a `simulation_results` object

`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)

Value

An object of class `simulation_results`

Examples

```
library(steps)
library(raster)
library(future)
plan(sequential)
```

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

pop_dynamics <- population_dynamics(change = growth(transition_matrix = egk_mat))

results <- simulation(landscape, pop_dynamics, timesteps = 10, replicates = 3)

# Test if object is of the type 'simulation_results'

is.simulation_results(results)

plot(results)

# Extract spatial components from a 'simulation_results' object

extract_spatial(results)
```

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

Usage

```
steps_stash
```

Format

An object of class `environment` of length 0.

Index

∗Topic **datasets**
egk, 3
steps, 15

all_dispersing, 2

carrying_capacity_dispersal, *2*
carrying_capacity_dispersal *(*all_dispersing*)*, 2
ceiling_density *(*population_density_dependence_functions*)*, 9
cellular_automata_dispersal *(*population_dispersal_functions*)*, 9
competition_density *(*modified_transition*)*, 7

disturbance *(*habitat_dynamics_functions*)*, 5

egk, 3
egk_fire *(*egk*)*, 3
egk_hab *(*egk*)*, 3
egk_k *(*egk*)*, 3
egk_mat *(*egk*)*, 3
egk_mat_stoch *(*egk*)*, 3
egk_pop *(*egk*)*, 3
egk_road *(*egk*)*, 3
egk_sf *(*egk*)*, 3
egk_sink *(*egk*)*, 3
egk_source *(*egk*)*, 3
exponential_dispersal_kernel, *4*, 4
extract_spatial *(*simulation*)*, 13

fast_dispersal *(*population_dispersal_functions*)*, 9
fire_effects *(*habitat_dynamics_functions*)*, 5

growth *(*population_change_functions*)*, 8

habitat_dynamics_functions, 5, *13*, *15*

is.landscape *(*landscape*)*, 6
is.population_dynamics *(*population_dynamics*)*, 11
is.simulation_results *(*simulation*)*, 13

kernel_dispersal *(*population_dispersal_functions*)*, 9

landscape, 6, *13*, *15*

modified_transition, 7

plot.simulation_results *(*simulation*)*, 13
population_change_functions, 8, *11*
population_density_dependence_functions, 9, *11*
population_dispersal_functions, 9, *11*
population_dynamics, 11, *13*, *15*
population_modification_functions, *11*, 12

simulation, *12*, 13, *15*
steps, 15
steps-package *(*steps*)*, 15
steps_stash *(*steps*)*, 15

transition_function, *8*
translocation *(*population_modification_functions*)*, 12