Package ‘rangr’

February 14, 2025

Type Package

Title Mechanistic Simulation of Species Range Dynamics

Version 1.0.5

Description Integrates population dynamics and dispersal into a mechanistic virtual species simulator. The package can be used to study the effects of environmental change on population growth and range shifts. It allows for simple and straightforward definition of population dynamics (including positive density dependence), extensive possibilities for defining dispersal kernels, and the ability to generate virtual ecologist data. Learn more about the 'rangr' at <https://docs.ropensci.org/rangr/>.

License MIT + file LICENSE

Imports methods, parallel, pbapply, grDevices, graphics, stats, utils, zoo, terra, assertthat

Suggests knitr, rmarkdown, testthat (>= 3.0.0), covr, bookdown

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Depends R (>= 3.5.0)

Config/testthat/edition 3

URL https://github.com/ropensci/rangr,
https://docs.ropensci.org/rangr/,
https://doi.org/10.1111/2041-210X.14475

BugReports https://github.com/ropensci/rangr/issues

NeedsCompilation no

Author Katarzyna Markowska [aut, cre],
Lechosław Kuczyński [aut],
Tad Dallas [rev],
Joanne Potts [rev]

Maintainer Katarzyna Markowska <katarzyna.markowska@amu.edu.pl>

Repository CRAN

Date/Publication 2025-02-14 14:30:02 UTC

Contents

`disp` *Simulating Dispersal*

Description

This function simulates dispersal for each grid cell by calculating the number of individuals dispersing out of the cell and the number of individuals dispersing into the cell.

Usage

```
disp(
  N_t,
  id,
  id_matrix,
  data_table,
  kernel,
  dens_dep,
  dlist,
  id_within,
  within_mask,
  border,
  planar,
  dist_resolution,
  max_dist,
  dist_bin,
  ncells_in_circle,
  cl = NULL
)
```

Arguments

`N_t`	integer matrix representing population numbers at a single time step; NA indicates cells outside the study area
`id`	`SpatRaster` object (of the same size as `N_t`) with cell identifiers
`id_matrix`	`id` in matrix format
`data_table`	matrix that contains information about all cells in current time points
`kernel`	function defining dispersal kernel
`dens_dep`	character vector of length 1 specifying if the probability of settling in a target grid cell is (case-sensitive, default `"K2N"`): • "none" - fully random, • "K" - proportional to the carrying capacity of a target cell, • "K2N" - density-dependent, i.e. proportional to the ratio of carrying capacity of a target cell to the number of individuals already present in a target cell
`dlist`	list with identifiers of target cells at a specified distance from a focal cell
`id_within`	integer vector with identifiers of cells inside the study area
`within_mask`	logical matrix that specifies boundaries of the study area
`border`	character vector of length 1 defining how to deal with borders (case-sensitive, default `"absorbing"`): • "reprising" - cells outside the study area are not allowed as targets for dispersal • "absorbing" - individuals that disperse outside the study area are removed from the population

`planar` logical vector of length 1; `TRUE` if input maps are planar rasters, `FALSE` if input maps are lon/lat rasters

`dist_resolution` integer vector of length 1; dimension of one side of one cell of `id`; in case of an irregular grid or lon/lat raster it is calculated during `initialisation`

`max_dist` distance (in the same units as used in the raster `id`) specifying the maximum range at which identifiers of target dispersal cells are determined in advance (see `initialise`)

`dist_bin` numeric vector of length 1 with value `>= 0`; in case of an irregular grid or lon/lat raster it is calculated during `initialisation`

`ncells_in_circle` numeric vector; number of cells on each distance

`cl` if simulation is done in parallel, the name of a cluster object created by `makeCluster`

Details

The function is used by `sim` internally and is not intended to be called by the user. The parameters for this function are passed from a `sim_data` object created by `initialise`.

Dispersal distance is expressed in original spatial units of the `SpatRaster` provided to the `sim` function (`n1_map` and `K_map`). However, it is internally converted to units of the simulation (i.e. the size of a single cell) by calculating `round(distance/resolution)`. If the selected dispersal distance is smaller than `resolution/2`, the individual does not disperse effectively and remains in the same cell. The dispersal rate (proportion of dispersing individuals) can be estimated from the dispersal kernel probability function by calculating the probability that the dispersal distance is greater than `resolution/2`.

Value

The function returns a list that contains two matrices:

`em` - emigration matrix with the number of individuals that dispersed from each cell

`im` - immigration matrix with the number of individuals that dispersed to each cell

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)

# disp
```

```
disp_output <- disp(
  N_t = sim_data$n1_map,
  id = unwrap(sim_data$id),
  id_matrix = as.matrix(unwrap(sim_data$id), wide = TRUE),
  data_table = sim_data$data_table,
  kernel = sim_data$kernel,
  dens_dep = sim_data$dens_dep,
  dlist = sim_data$dlist,
  id_within = sim_data$id_within,
  within_mask = sim_data$within_mask,
  border = sim_data$border,
  planar = sim_data$planar,
  dist_resolution = sim_data$dist_resolution,
  max_dist = sim_data$max_dist,
  dist_bin = sim_data$dist_bin,
  ncells_in_circle = sim_data$ncells_in_circle
)

# immigration and emigration matrices
names(disp_output)
```

`get_observations` *Observation Process*

Description

This function simulates an observation process. It accepts the `sim_results` object, which is generated by the `sim` function, and applies the virtual ecologist approach on the `N_map` component of the object. The function returns a `data.frame` with the 'observed' abundances.

Usage

```
get_observations(
  sim_data,
  sim_results,
  type = c("random_one_layer", "random_all_layers", "from_data", "monitoring_based"),
  obs_error = c("rlnorm", "rbinom"),
  obs_error_param = NULL,
  ...
)
```

Arguments

`sim_data`	`sim_data` object from `initialise` containing simulation parameters
`sim_results`	`sim_results` object; returned by `sim` function
`type`	character vector of length 1; describes the sampling type (case-sensitive):

- "random_one_layer" - random selection of cells for which abundances are sampled; the same set of selected cells is used across all time steps.
- "random_all_layers" - random selection of cells for which abundances are sampled; a new set of cells is selected for each time step.
- "from_data" - user-defined selection of cells for which abundances are sampled; the user is required to provide a `data.frame` containing three columns: "x", "y" and "time_step".
- "monitoring_based" - user-defined selection of cells for which abundances are sampled; the user is required to provide a matrix object with two columns: "x" and "y"; the abundance from given cell is sampled by different virtual observers in different time steps; a geometric distribution (rgeom) is employed to define whether a survey will be conducted by the same observer for several years or not conducted at all.

`obs_error` character vector of length 1; type of the distribution that defines the observation process: "rlnorm" (the log normal distribution) or "rbinom" (the binomial distribution)

`obs_error_param` numeric vector of length 1; standard deviation (on a log scale) of the random noise in observation process generated from the log-normal distribution (rlnorm) or probability of detection (success) when the binomial distribution ("rbinom") is used.

`...` other necessary internal parameters:

- `prop`
  numeric vector of length 1; proportion of cells to be sampled (default `prop = 0.1`); used when `type = "random_one_layer" or "random_all_layers"`,
- `points`
  `data.frame` or `matrix` with 3 numeric columns named "x", "y", and "time_step" containing coordinates and time steps from which observations should be obtained; used when `type = "from_data"`,
- `cells_coords`
  `data.frame` or `matrix` with 2 columns named "x" and "y"; survey plots coordinates; used when `type = "monitoring_based"`
- `prob`
  numeric vector of length 1; a parameter defining the shape of rgeom distribution; defines whether an observation will be made by the same observer for several years, and whether it will not be made at all (default `prob = 0.3`); used when `type = "monitoring_based"`
- `progress_bar`
  logical vector of length 1; determines if a progress bar for observation process should be displayed (default `progress_bar = FALSE`); used when `type = "monitoring_based"`

Value

`data.frame` object with geographic coordinates, time steps, estimated abundance, observation error (if `obs_error_param` is provided), and observer identifiers (if `type = "monitoring_based"`). If `type = "from_data"`, returned object is sorted in the same order as the input `points`.

Examples

```
library(terra)
n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

# prepare data
sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)

sim_1 <- sim(obj = sim_data, time = 110, burn = 10)

# 1. random_one_layer
sample1 <- get_observations(
  sim_data,
  sim_1,
  type = "random_one_layer",
  prop = 0.1
)

# 2. random_all_layers
sample2 <- get_observations(
  sim_data,
  sim_1,
  type = "random_all_layers",
  prop = 0.15
)

# 3. from_data
sample3 <- get_observations(
  sim_data,
  sim_1,
  type = "from_data",
  points = observations_points
)

# 4. monitoring_based
# define observations sites
all_points <- xyFromCell(unwrap(sim_data$id), cells(unwrap(sim_data$K_map)))
sample_idx <- sample(1:nrow(all_points), size = 20)
sample_points <- all_points[sample_idx, ]

sample4 <- get_observations(
  sim_data,
  sim_1,
  type = "monitoring_based",
  cells_coords = sample_points,
  prob = 0.3,
```

```
  progress_bar = TRUE
)

# 5. noise "rlnorm"
sample5 <- get_observations(sim_data,
  sim_1,
  type = "random_one_layer",
  obs_error = "rlnorm",
  obs_error_param = log(1.2)
)

# 6. noise "rbinom"
sample6 <- get_observations(sim_data,
  sim_1,
  type = "random_one_layer",
  obs_error = "rbinom",
  obs_error_param = 0.8
)
```

`growth` *Population Growth Functions*

Description

Population growth functions are used during simulation conducted by the `sim` function. The user is required to specify the name of a growth function while initialising the `sim_data` object using `initialise`.

Usage

```
exponential(x, r, ...)

ricker(x, r, K, A = NA)

gompertz(x, r, K, A = NA)
```

Arguments

`x`	number of individuals
`r`	intrinsic population growth rate
`...`	not used, added for compatibility reasons
`K`	carrying capacity
`A`	coefficient of Allee effect (A <= 0: weak, A > 0: strong, NA: none)

Details

`x` can be a vector, matrix, `SpatRaster` or any other `R` object for which basic arithmetic operations produce valid results. These functions are intended to be used in the `sim` function, where `x` is a matrix of the same dimensions as the `SpatRaster` object specified in `n1_map` parameter.

Value

Object of the same dimensions as `x` that contains expected number of individuals in the next time step.

References

Boukal, D. S., & Berec, L. (2002). Single-species models of the Allee effect: extinction boundaries, sex ratios and mate encounters. Journal of Theoretical Biology, 218(3), 375-394. doi:10.1006/jtbi.2002.3084

Gompertz, B. (1825) On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contigencies. Philosophical Transactions of the Royal Society of London, 115, 513-583. doi:10.1098/rstl.1825.0026

Ricker, W.E. (1954) Stock and Recruitment. Journal of the Fisheries Research Board of Canada, 11, 559-623. doi:10.1139/f54039

Hostetler, J.A. and Chandler, R.B. (2015), Improved state-space models for inference about spatial and temporal variation in abundance from count data. Ecology, 96: 1713-1723. doi:10.1890/14-1487.1

Courchamp, F., L. Berec and J. Gascoigne. 2008. Allee Effects in Ecology and Conservation. Oxford University Press, New York. 256 pp. ISBN 978-0-19-857030-1

Examples

```
x <- 1:10
exponential(x, r = 0.4)

ricker(x, r = 2, K = 5)
ricker(x, r = 2, K = 5, A = -5)

gompertz(x, r = 1.2, K = 5)
gompertz(x, r = 1.2, K = 5, A = 5)
```

`initialise` *Prepare Data Required To Perform A Simulation*

Description

This function generates a `sim_data` object containing all the necessary information required to run a simulation by the `sim` function. The input maps (`n1_map` and `K_map`) can be in the Cartesian or longitude/latitude coordinate system.

Usage

```
initialise(
  n1_map,
  K_map,
  K_sd = 0,
  r,
  r_sd = 0,
  growth = "gompertz",
  A = NA,
  dens_dep = c("K2N", "K", "none"),
  border = c("reprising", "absorbing"),
  kernel_fun = "rexp",
  ...,
  max_dist = NA,
  calculate_dist = TRUE,
  dlist = NULL,
  progress_bar = TRUE,
  quiet = FALSE
)
```

Arguments

`n1_map`	`SpatRaster` object with one layer; population numbers in every grid cell at the first time step
`K_map`	`SpatRaster` object with one layer; carrying capacity map (if K is constant across time) or maps (if K is time-varying)
`K_sd`	numeric vector of length 1 with value `>= 0` (default 0); this parameter can be used if additional environmental stochasticity is required; if `K_sd` > `0`, random numbers are generated from a log-normal distribution with the mean `K_map` and standard deviation `K_sd`
`r`	numeric vector of length 1; intrinsic population growth rate
`r_sd`	numeric vector of length 1 with value `>= 0` (default `0`); if additional demographic stochasticity is required, `r_sd` > `0` is the standard deviation for a normal distribution around `r` (defined for each time step)
`growth`	character vector of length 1; the name of a population growth function, either defined in `growth` or provided by the user (case-sensitive, default `"gompertz"`)
`A`	numeric vector of length 1; strength of the Allee effect (see the `growth` function)
`dens_dep`	character vector of length 1 specifying if the probability of settling in a target grid cell is (case-sensitive, default `"K2N"`): • "none" - fully random, • "K" - proportional to the carrying capacity of a target cell, • "K2N" - density-dependent, i.e. proportional to the ratio of carrying capacity of a target cell to the number of individuals already present in a target cell
`border`	character vector of length 1 defining how to deal with borders (case-sensitive, default `"absorbing"`):

- "reprising" - cells outside the study area are not allowed as targets for dispersal
- "absorbing" - individuals that disperse outside the study area are removed from the population

kernel_fun	character vector of length 1; name of a random number generation function defining a dispersal kernel (case-sensitive, default "rexp")
...	any parameters required by kernel_fun
max_dist	numeric vector of length 1; maximum distance of dispersal to pre-calculate target cells
calculate_dist	logical vector of length 1; determines if target cells will be precalculated
dlist	list; target cells at a specified distance calculated for every cell within the study area
progress_bar	logical vector of length 1; determines if progress bar for calculating distances should be displayed
quiet	logical vector of length 1; determines if messages should be displayed

Details

The most time-consuming part of computations performed by the sim function is the simulation of dispersal. To speed it up, a list containing indexes of target cells at a specified distance from a focal cell is calculated in advance and stored in a dlist slot. The max_dist parameter sets the maximum distance at which this pre-calculation is performed. If max_dist is NULL, it is set to 0.99 quantile from the kernel_fun. All distance calculations are always based on metres if the input maps are latitude/longitude. For planar input maps, distances are calculated in map units, which are typically metres, but check the crs() if in doubt.

If the input maps are in the Cartesian coordinate system and the grid cells are squares, then the distances between cells are calculated using the distance function from the terra package. These distances are later divided by the resolution of the input maps.

For input maps with grid cells in shapes other than squares (e.g. with rectangular cells or longitude/latitude coordinate system), the distance resolution is calculated by finding the shortest distance between each "queen" type neighbor. All distances calculated by the distance function are further divided by this distance resolution. To avoid discontinuities in the distances at which the target cells are located, an additional parameter dist_bin is calculated as half of the maximum distance between each "queen" type neighbour. It is used to expand the distances at which target cells are located from a single number to a range.

NA in the input maps represents cells outside the study area.

The K_get_interpolation function can be used to prepare K_map that changes over time. This may be useful, when simulating environmental change or exploring the effects of ecological disturbances.

Value

Object of class sim_data which inherits from list. This object contains all necessary information to perform a simulation using sim function.

References

Hijmans R (2024). terra: Spatial Data Analysis. R package version 1.7-81, https://rspatial.github.io/terra/, https://rspatial.org/

Solymos P, Zawadzki Z (2023). pbapply: Adding Progress Bar to '*apply' Functions. R package version 1.7-2, https://CRAN.R-project.org/package=pbapply.

See Also

update

Examples

```
# input maps
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))
K_small_changing <- rast(system.file("input_maps/K_small_changing.tif",
                         package = "rangr"))
n1_small_lon_lat <- rast(system.file("input_maps/n1_small_lon_lat.tif", package = "rangr"))
K_small_lon_lat <- rast(system.file("input_maps/K_small_lon_lat.tif", package = "rangr"))

# basic example
sim_data_1 <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)

# example with changing environment
K_interpolated <- K_get_interpolation(
  K_small_changing,
  K_time_points = c(1, 25, 50)
)

sim_data_2 <- initialise(
  n1_map = n1_small,
  K_map = K_interpolated,
  r = log(2),
  rate = 1 / 1e3
)

# example with lon/lat rasters
sim_data_3 <- initialise(
  n1_map = n1_small_lon_lat,
  K_map = K_small_lon_lat,
  r = log(2),
  rate = 1 / 1e3
)
```

```
# example without progress bar and messages
sim_data_4 <- initialise(
  n1_map = n1_small, K_map = K_small, K_sd = 0.1, r = log(5),
  r_sd = 4, growth = "ricker", rate = 1 / 200,
  max_dist = 5000, dens_dep = "K2N", progress_bar = FALSE, quiet = TRUE
)
```

`K_big.tif` *Example Of Carrying Capacity Map (Big)*

Description

`SpatRaster` object that can be used as a carrying capacity map to `initialise` data necessary to perform a simulation with the `sim` function. This map is compatible with `n1_big.tif`.

Format

`SpatRaster` object with 100 rows and 100 columns containing integer values 0-25 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_big.tif", package = "rangr")
```

`K_big_lon_lat.tif` *Example Of Carrying Capacity Map (Big)*

Description

`SpatRaster` object representing a carrying capacity map projected to WGS 84 (CRS84) from the original raster `K_big`. This map can be used as a carrying capacity map to `initialise` data necessary to perform a simulation with the `sim` function. It is compatible with the `n1_big_lon_lat.tif` raster.

Format

`SpatRaster` object with 74 rows and 125 columns containing integer values 0-25 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_big_lon_lat.tif", package = "rangr")
```

K_get_interpolation *Prepare Time-Varying Carrying Capacity Maps*

Description

This function linearly interpolates values in a series of carrying capacity maps.

Usage

```
K_get_interpolation(K_map, K_time_points = NULL, time = NULL)
```

Arguments

`K_map`	`SpatRaster` object with carrying capacity maps for each `K_time_points`
`K_time_points`	integer vector; time for each layer in `K_map`, should contain unique values
`time`	integer vector of length 1; number of total time steps required (this is defined when evoking the function `sim`).

Details

To simulate dynamic environmental scenarios (e.g. climate change, land use change, ecological disturbance, etc.) one needs to provide time-varying carrying capacity maps.

Either `K_time_points` or the `time` parameter is needed to perform interpolation. If the interpolation should be calculated between two carrying capacity maps, there is no need to pass the time points, because 1 will be set as the starting time point and `time` will be used as the ending point. On the other hand, in the absence of the `time` argument, the maximum element of `K_time_points` is considered to be the ending point for the interpolation.

Value

`SpatRaster` object with number of layers equal to `time`.

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small_changing <- rast(system.file("input_maps/K_small_changing.tif",
package = "rangr"))

K_interpolated_01 <- K_get_interpolation(
  K_small_changing,
  K_time_points = c(1, 10, 15)
)

K_two_layers <- subset(
  K_small_changing,
  c(1, 2)
)
K_interpolated_02 <- K_get_interpolation(
  K_two_layers,
  time = 15
)
```

K_small.tif *Example Of Carrying Capacity Map (Small)*

Description

SpatRaster object that can be used a carrying capacity map to initialise data necessary to perform a simulation with the sim function. This map is compatible with n1_small.tif.

Format

SpatRaster object with 15 rows and 10 columns containing integer values 0-100 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_small.tif", package = "rangr")
```

K_small_changing.tif *Example Of Changing Carrying Capacity Maps (Small)*

Description

SpatRaster object that can be used as carrying capacity maps to initialise data necessary to perform a simulation with the sim function. To utilise these maps in initialise the user first must use K_get_interpolation to generate a map for every time step of the simulation. These maps are compatible with n1_small.tif. Each subsequent map contains a virtual environment with greater carrying capacity than the previous one.

Format

SpatRaster object with 3 layers, each has 15 rows and 10 columns containing integer values 0-170 and NA's that indicates unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_small_changing.tif", package = "rangr")
```

K_small_changing_lon_lat.tif *Example Of Changing Carrying Capacity Maps (Small)*

Description

SpatRaster object representing changing carrying capacity maps projected to WGS 84 (CRS84) from the original raster K_small_changing. These maps can be used as carrying capacity maps to initialise data necessary to perform a simulation with the sim function. To utilise these maps in initialise the user must first use K_get_interpolation to generate a map for every time step of the simulation. These maps are compatible with the n1_small_lon_lat.tif raster.

Format

SpatRaster object with 3 layers, each having 12 rows and 14 columns containing integer values 0-170 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_small_changing_lon_lat.tif", package = "rangr")
```

`K_small_lon_lat.tif` *Example Of Carrying Capacity Map (Small)*

Description

`SpatRaster` object that represents a carrying capacity map projected to WGS 84 (CRS84) from the original raster `K_small`. This map can be used as a carrying capacity map to `initialise` data necessary to perform a simulation with the `sim` function. It is compatible with the `n1_small_lon_lat.tif` raster.

Format

`SpatRaster` object with 12 rows and 14 columns containing integer values 0-100 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example (using spatial autocorrelation).

Examples

```
system.file("input_maps/K_small_lon_lat.tif", package = "rangr")
```

`n1_big.tif` *Example Of Abundance Map At First Time Step Of The Simulation (Big)*

Description

`SpatRaster` object that can be used a as simulation starting point to `initialise` data necessary to perform a simulation with the `sim` function. This map is compatible with `K_big.tif` map.

Format

`SpatRaster` object with 100 rows and 100 columns containing integer values 0-50 and NA's that indicates unsuitable areas.

Source

Data generated in-house to serve as an example.

Examples

```
system.file("input_maps/n1_big.tif", package = "rangr")
```

---

n1_big_lon_lat.tif *Example Of Abundance Map At First Time Step Of The Simulation (Big)*

Description

SpatRaster object representing an abundance map at the first time step of the simulation projected to WGS 84 (CRS84) from the original raster n1_big. This map can be used as a simulation starting point to initialise data necessary to perform a simulation with the sim function. It is compatible with the K_big_lon_lat.tif map.

Format

SpatRaster object with 74 rows and 125 columns containing integer values 0-50 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example.

Examples

```
system.file("input_maps/n1_big_lon_lat.tif", package = "rangr")
```

---

n1_small.tif *Example Of Abundance Map At First Time Step Of The Simulation (Small)*

Description

SpatRaster object that can be used a as simulation starting point to initialise data necessary to perform a simulation with the sim function. This map is compatible with K_small.tif and K_small_changing.tif maps.

Format

SpatRaster object with 15 rows and 10 columns containing integer values 0-10 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example.

Examples

```
system.file("input_maps/n1_small.tif", package = "rangr")
```

`n1_small_lon_lat.tif` *Example Of Abundance Map At First Time Step Of The Simulation (Small)*

Description

`SpatRaster` object representing an abundance map at the first time step of the simulation projected to WGS 84 (CRS84) from the original raster `n1_small`. This map can be used as a simulation starting point to `initialise` data necessary to perform a simulation with the `sim` function. It is compatible with the `K_small_lon_lat.tif` and `K_small_changing_lon_lat.tif` maps.

Format

`SpatRaster` object with 12 rows and 14 columns containing integer values 0-10 and NA's indicating unsuitable areas.

Source

Data generated in-house to serve as an example.

Examples

```
system.file("input_maps/n1_small_lon_lat.tif", package = "rangr")
```

`observations_points` *Example Of Observation Points List*

Description

A `data.frame` containing a sample input data to the function `get_observations` when type argument is set to "from_file". This data is compatible with `n1_small.tif`, `K_small.tif` and `K_small_changing.tif` maps.

Usage

```
observations_points
```

Format

A data frame with 1500 rows and 3 variables:

x x coordinate

y y coordinate

time_step time_step at which the abundances should be observed

Source

Data generated in-house to serve as an example

---

plot.sim_results *Plot* sim_results *Object*

Description

Plots abundances obtained during simulation.

Usage

```
## S3 method for class 'sim_results'
plot(x, template = NULL, time_points = NULL, range, type, ...)
```

Arguments

x	sim_results object; returned by sim
template	SpatRaster object; can be used as a template to create returned object
time_points	numeric vector; specifies points in time from which plots will be generated
range	numeric vector of length 2; range of values to be used for the legend (if type = "continuous"), which by default is calculated from the N_map slot of sim_result object
type	character vector of length 1; type of map: "continuous" (default), "classes" or "interval" (case-sensitive)
...	further arguments passed to terra::plot

Value

SpatRaster object with as many layers as the length of time_points parameter

Examples

```
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
sim_res <- sim(sim_data, time = 10)
plot(sim_res)
plot(sim_res, template = n1_small, time_points = c(1, 10))

# plot specific area
plot(sim_res, xlim = c(4, 10), ylim = c(0, 10))
plot(sim_res, ext = c(4, 10, 0, 10))
plot(sim_res, template = n1_small, ext = c(274000, 280000, 610000, 620000))
```

---

print.sim_data *Print* sim_data *Object*

Description

Print sim_data Object

Usage

```
## S3 method for class 'sim_data'
print(x, ...)
```

Arguments

x	sim_data object; returned by the initialise function
...	further arguments passed to or from other methods; currently none specified

Value

sim_data object is invisibly returned (the x param)

Examples

```
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
print(sim_data)
```

`print.sim_results` *Print* `sim_results` *Object*

Description

Print `sim_results` Object

Usage

```
## S3 method for class 'sim_results'
print(x, ...)
```

Arguments

`x`	`sim_results` object; returned by the `sim` function
`...`	further arguments passed to or from other methods; none specified

Value

`sim_results` object is invisibly returned (the `x` param)

Examples

```
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
```

```
sim_res <- sim(obj = sim_data, time = 20, burn = 5)
print(sim_res)
```

---

print.summary.sim_data

Print summary.sim_data *Object*

Description

Print summary.sim_data Object

Usage

```
## S3 method for class 'summary.sim_data'
print(x, ...)
```

Arguments

x	summary.sim_data object; returned by summary.sim_data function
...	further arguments passed to or from other methods; currently none specified

Value

None

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
summary_sim_data <- summary(sim_data)
print(summary_sim_data)
```

print.summary.sim_results *Print* summary.sim_results *Object*

Description

Print summary.sim_results Object

Usage

```
## S3 method for class 'summary.sim_results'
print(x, ...)
```

Arguments

x	summary.sim_results object; returned by summary.sim_results function
...	further arguments passed to or from other methods; currently none specified

Value

None

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
sim_results <- sim(sim_data, time = 10)
summary_sim_results <- summary(sim_results)
print(summary_sim_results)
```

sim *Mechanistic Metapopulation Simulator*

Description

This function simulates population growth and dispersal providing a given environmental scenario. All parameters for the simulation must be set in advance using initialise.

Usage

```
sim(
  obj,
  time,
  burn = 0,
  return_mu = FALSE,
  cl = NULL,
  progress_bar = TRUE,
  quiet = FALSE
)
```

Arguments

obj	sim_data object created by initialise containing all simulation parameters and necessary data
time	positive integer vector of length 1; number of time steps simulated
burn	positive integer vector of length 1; the number of burn-in time steps that are discarded from the output
return_mu	logical vector of length 1; if TRUE demographic process return expected values; if FALSE the rpois function should be used
cl	an optional cluster object created by makeCluster needed for parallel calculations
progress_bar	logical vector of length 1 determines if progress bar for simulation should be displayed
quiet	logical vector of length 1; determines if warnings should be displayed

Details

This is the main simulation module. It takes the sim_data object prepared by initialise and runs simulation for a given number of time steps. The initial (specified by the burn parameter) time steps are skipped and discarded from the output. Computations can be done in parallel if the name of a cluster created by makeCluster is provided.

Generally, at each time step, simulation consists of two phases: local dynamics and dispersal.

Local dynamics (which connects habitat patches in time) is defined by the function growth. This parameter is specified while creating the obj using initialise, but can be later modified by using the update function. Population growth can be either exponential or density-dependent, and the regulation is implemented by the use of Gompertz or Ricker models (with a possibility of providing any other, user defined function). For every cell, the expected population density during the next time step is calculated from the corresponding number of individuals currently present in this cell, and the actual number of individuals is set by drawing a random number from the Poisson distribution using this expected value. This procedure introduces a realistic randomness, however additional levels of random variability can be incorporated by providing parameters of both demographic and environmental stochasticity while specifying the `sim_data` object using the `initialise` function (parameters `r_sd` and `K_sd`, respectively).

To simulate dispersal (which connects habitat patches in space), for each individual in a given cell, a dispersal distance is randomly drawn from the dispersal kernel density function. Then, each individual is translocated to a randomly chosen cell at this distance apart from the current location. For more details, see the `disp` function.

Value

This function returns an object of class `sim_results` which is a list containing the following components:

- `extinction` - `TRUE` if population is extinct or `FALSE` otherwise
- `simulated_time` - number of simulated time steps without the burn-in ones
- `N_map` - 3-dimensional array representing spatio-temporal variability in population numbers. The first two dimensions correspond to the spatial aspect of the output and the third dimension represents time.

In case of a total extinction, a simulation is stopped before reaching the specified number of time steps. If the population died out before reaching the `burn` threshold, then nothing can be returned and an error occurs.

References

Solymos P, Zawadzki Z (2023). pbapply: Adding Progress Bar to '*apply' Functions. R package version 1.7-2, https://CRAN.R-project.org/package=pbapply.

See Also

get_observations

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
```

```
  rate = 1 / 1e3
)

# simulation
sim_1 <- sim(obj = sim_data, time = 20)

# simulation with burned time steps
sim_2 <- sim(obj = sim_data, time = 20, burn = 10)

# example with parallelization
library(parallel)
cl <- makeCluster(2)

# parallelized simulation
sim_3 <- sim(obj = sim_data, time = 20, cl = cl)
stopCluster(cl)


# visualisation
plot(
  sim_1,
  time_points = 20,
  template = sim_data$K_map
)

plot(
  sim_1,
  time_points = c(1, 5, 10, 20),
  template = sim_data$K_map
)

plot(
  sim_1,
  template = sim_data$K_map
)
```

`subset.sim_results` *Subset of Given Time Points from* `sim_results` *Object*

Description

This function creates a subset of given time points from the `sim_results` object.

Usage

```
## S3 method for class 'sim_results'
subset(x, from = NULL, time_points = NULL, ...)
```

Arguments

x	sim_results object; returned by the sim function
from	numeric vector of length 1; indicates the starting time point from which all time point should be kept
time_points	numeric vector; indicates all time points to keep
...	further arguments to be passed to or from other methods

Details

Either from or time_points argument has to be specified. Time point passed by the from argument will be set as a cutoff point and all abundances for previous time points will be discarded.

Value

sim_results object with only selected time_points present in the N_map slot

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n = n1_small,
  r = log(2),
  K_map = K_small,
  max_dist = 1000,
  rate = 1 / 1e3
)

sim_results <- sim(sim_data, time = 10)
summary(sim_results)

sim_results_cropped <- subset(sim_results, time_points = c(1:2))
summary(sim_results_cropped)
```

---

summary.sim_data *Summary Of* sim_data *Object*

Description

Summary Of sim_data Object

Usage

```
## S3 method for class 'sim_data'
summary(object, ...)
```

Arguments

`object`	`sim_data` object; returned by `initialise` function
`...`	further arguments passed to or from other methods; currently none specified

Value

`summary.sim_data` object

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
summary(sim_data)
```

`summary.sim_results` *Summary Of* `sim_results` *Object*

Description

Summary Of `sim_results` Object

Usage

```
## S3 method for class 'sim_results'
summary(object, ...)
```

Arguments

`object`	`sim_results` object; returned by `sim` function
`...`	further arguments passed to or from other methods; none specified

Value

`summary.sim_results` object

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)

# simulation
sim_results <- sim(sim_data, time = 10)
summary(sim_results)
```

---

to_rast — *Transformation* `sim_results` *To Raster*

Description

This function transforms selected subset of abundance matrices from `sim_results` into `SpatRaster` object. Layers are specified by `time_points`, which can be one or multiple points in time.

Usage

```
to_rast(sim_results, time_points = sim_results$simulated_time, template = NULL)
```

Arguments

`sim_results`	`sim_results` object created by `sim`
`time_points`	numeric vector of length 1 or more; specifies points in time from which `SpatRaster` will be created - as default the last year of simulation; if `length(time_points)` > `0` `SpatRaster` will be returned with layers for each element of `time_points`
`template`	`SpatRaster` object; can be used as a template to create returned object

Value

`SpatRaster` based on `sim_results` object with layers corresponding to `time_points`.

References

Hijmans R (2024). terra: Spatial Data Analysis. R package version 1.7-81, https://rspatial.github.io/terra/, https://rspatial.org/

Examples

```
# data preparation
library(terra)

n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)

# simulation
sim_1 <- sim(obj = sim_data, time = 100)

# raster construction
my_rast <- to_rast(
  sim_1,
  time_points = c(1, 10, 20, 100),
  template = sim_data$K_map
)

# visualization
plot(my_rast, range = range(sim_1$N_map, na.rm = TRUE))
```

`update.sim_data` *Update* `sim_data` *Object*

Description

This function updates a `sim_data` object.

Usage

```
## S3 method for class 'sim_data'
update(object, ..., evaluate = TRUE)
```

Arguments

object	sim_data object; returned by initialise function
...	further arguments passed to or from other methods; currently none specified
evaluate	logical vector of length 1; if TRUE evaluates the new call, otherwise returns the call

Value

If evaluate = TRUE then the updated sim_data object, otherwise the updated call.

Examples

```
# data preparation
library(terra)
n1_small <- rast(system.file("input_maps/n1_small.tif", package = "rangr"))
K_small <- rast(system.file("input_maps/K_small.tif", package = "rangr"))

sim_data_1 <- initialise(
  n1_map = n1_small,
  K_map = K_small,
  r = log(2),
  rate = 1 / 1e3
)
summary(sim_data_1)

sim_data_2 <- update(sim_data_1, max_dist = 3000)
summary(sim_data_2)
```

Index