Package: gcube (via r-universe)

February 13, 2025

Title Simulating Biodiversity Data Cubes

Version 1.1.2

Description This R package provides a simulation framework for biodiversity data cubes. This can start from simulating multiple species distributed in a landscape over a temporal scope. In a second phase, the simulation of a variety of observation processes and effort can generate actual occurrence datasets. Based on their (simulated) spatial uncertainty, occurrences can then be designated to a grid to form a data cube.

License MIT + file LICENSE

URL https://github.com/b-cubed-eu/gcube, https://b-cubed-eu.github.io/gcube/, https://doi.org/10.5281/zenodo.14038996

BugReports https://github.com/b-cubed-eu/gcube/issues

Imports assertthat, dplyr, gstat, methods, mnormt, purrr, rlang, sf, stats, terra, tidyr, vegan

Suggests ggExtra, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0), tidyterra

Config/checklist/communities b3; inbo

Config/checklist/keywords simulation; data cubes; B-Cubed; biodiversity; Monte-Carlo

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libicu-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://b-cubed-eu.r-universe.dev

RemoteUrl https://github.com/b-cubed-eu/gcube

RemoteRef HEAD

RemoteSha e014771fb59fbeeaf3732a6d318ca7d9ec7f1303

Contents

`add_coordinate_uncertainty` *Add coordinate uncertainty to observations*

Description

This function adds a column to the input dataframe or sf object containing the coordinate uncertainty for each observation, measured in meters.

Usage

```
add_coordinate_uncertainty(observations, coords_uncertainty_meters = 25)
```

Arguments

`observations`	An sf object with POINT geometry or a simple dataframe representing the observations. This object contains the observation points to which the coordinate uncertainty will be added.
`coords_uncertainty_meters`	A numeric value or a vector of numeric values representing the coordinate uncertainty (in meters) associated with each observation. If a single numeric value is provided, it will be applied to all observations. If a numeric vector is provided, it must be the same length as the number of observations.

Value

The input data frame or an sf object with POINT geometry, with an additional column named `coordinateUncertaintyInMeters` that contains the coordinate uncertainty values in meters.

See Also

Other main: `filter_observations()`, `grid_designation()`, `sample_observations()`, `simulate_occurrences()`

Examples

```
# Create dataframe with sampling status column
observations_data <- data.frame(
    time_point = 1,
    sampling_prob = seq(0.5, 1, 0.1)
  )

# provide a fixed uncertainty for all points
add_coordinate_uncertainty(
  observations_data,
  coords_uncertainty_meters = 1000
 )

# add variability in uncertainty. For example, using gamma distribution
uncertainty_vec <- seq(50, 100, 10)

add_coordinate_uncertainty(
  observations_data,
  coords_uncertainty_meters = uncertainty_vec
)
```

---

apply_manual_sampling_bias

Apply manual sampling bias to occurrences via a grid

Description

This function adds a sampling bias weight column to an sf object containing occurrences. The sampling probabilities are based on bias weights within each cell of a provided grid layer.

Usage

```
apply_manual_sampling_bias(occurrences_sf, bias_weights)
```

Arguments

`occurrences_sf`	An sf object with POINT geometry representing the occurrences.
`bias_weights`	An `sf` object with POLYGON geometry representing the grid with bias weights. This sf object should contain a `bias_weight` column and a `geometry` column. Higher weights indicate a higher probability of sampling. Weights must be numeric values between 0 and 1 or positive integers, which will be rescaled to values between 0 and 1.

Value

An sf object with POINT geometry that includes a `bias_weight` column containing the sampling probabilities based on the sampling bias.

See Also

Other detection: `apply_polygon_sampling_bias()`

Examples

```
# Load packages
library(sf)
library(dplyr)
library(ggplot2)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

# Get occurrence points
occurrences_sf <- simulate_occurrences(plgn)

# Create grid with bias weights
grid <- st_make_grid(
    plgn,
    n = c(10, 10),
    square = TRUE) %>%
  st_sf()
grid$bias_weight <- runif(nrow(grid), min = 0, max = 1)

# Calculate occurrence bias
occurrence_bias <- apply_manual_sampling_bias(occurrences_sf, grid)
```

```
occurrence_bias

# Visualise where the bias is
ggplot() +
  geom_sf(data = plgn) +
  geom_sf(data = grid, alpha = 0) +
  geom_sf(data = occurrence_bias, aes(colour = bias_weight)) +
  geom_sf_text(data = grid, aes(label = round(bias_weight, 2))) +
  theme_minimal()
```

`apply_polygon_sampling_bias` *Apply sampling bias to occurrences via a polygon*

Description

This function adds a sampling bias weight column to an `sf` object containing occurrences based on a given polygonal area. The bias is determined by the specified bias strength, which adjusts the probability of sampling within the polygonal area.

Usage

```
apply_polygon_sampling_bias(occurrences_sf, bias_area, bias_strength = 1)
```

Arguments

`occurrences_sf`	An sf object with POINT geometry representing the occurrences.
`bias_area`	An sf object with POLYGON geometry specifying the area where sampling will be biased.
`bias_strength`	A positive numeric value that represents the strength of the bias to be applied within the `bias_area`. Values greater than 1 will increase the sampling probability within the polygon relative to outside (oversampling), while values between 0 and 1 will decrease it (undersampling). For instance, a value of 50 will make the probability 50 times higher within the `bias_area` compared to outside, whereas a value of 0.5 will make it half as likely.

Value

An sf object with POINT geometry that includes a `bias_weight` column containing the sampling probabilities based on the bias area and strength.

See Also

Other detection: `apply_manual_sampling_bias()`

Examples

```
# Load packages
library(sf)
library(dplyr)
library(ggplot2)

# Simulate some occurrence data with coordinates and time points
num_points <- 10
occurrences <- data.frame(
  lon = runif(num_points, min = -180, max = 180),
  lat = runif(num_points, min = -90, max = 90),
  time_point = 1
  )

# Convert the occurrence data to an sf object
occurrences_sf <- st_as_sf(occurrences, coords = c("lon", "lat"))

# Create bias_area polygon overlapping at least two of the points
selected_observations <- st_union(occurrences_sf[2:3,])
bias_area <- st_convex_hull(selected_observations) %>%
  st_buffer(dist = 50) %>%
  st_as_sf()

occurrence_bias_sf <- apply_polygon_sampling_bias(
  occurrences_sf,
  bias_area,
  bias_strength = 2)
occurrence_bias_sf

# Visualise where the bias is
occurrence_bias_sf %>%
  mutate(bias_weight = as.factor(round(bias_weight, 3))) %>%
  ggplot() +
    geom_sf(data = bias_area) +
    geom_sf(aes(colour = bias_weight)) +
    theme_minimal()
```

---

create_spatial_pattern

Create spatial pattern within a polygon

Description

This function creates a raster with a spatial pattern for the area of a polygon.

Usage

```
create_spatial_pattern(
  polygon,
```

```
  resolution,
  spatial_pattern = c("random", "clustered"),
  seed = NA,
  n_sim = 1
)
```

Arguments

`polygon`	An sf object with POLYGON geometry.
`resolution`	A numeric value defining the resolution of the raster cells.
`spatial_pattern`	Specifies the desired spatial pattern. It can be a character string (`"random"` or `"clustered"`) or a numeric value 1 (1 means random distribution, larger values indicate more clustering). The default is `"random"`. `"clustered"` corresponds to a value of 10. See Details.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.
`n_sim`	Number of simulations. Each simulation is a different layer in the raster. Default is 1.

Details

The `spatial_pattern` argument changes the range parameter of the spherical variogram model. `spatial_pattern = 1` means the range has the same size as the grid cell, which is defined in the `resolution` argument. The function `gstat::vgm()` is used to implement the spherical variogram model.

Value

An object of class SpatRaster with a spatial pattern for the area of the given polygon with `n_sim` layers `sampling_p'n_sim'` containing the sampling probabilities from the raster grid for each simulation.

See Also

`gstat::vgm()` and its `range` argument

Other occurrence: `sample_occurrences_from_raster()`, `simulate_random_walk()`, `simulate_timeseries()`

Examples

```
# Load packages
library(sf)
library(ggplot2)
library(tidyterra)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))
```

```
# 1. Random spatial pattern
rs_pattern_random <- create_spatial_pattern(
  polygon = plgn,
  resolution = 0.1,
  spatial_pattern = "random",
  seed = 123)

ggplot() +
  geom_spatraster(data = rs_pattern_random) +
  scale_fill_continuous(type = "viridis") +
  theme_minimal()

# 2. Clustered spatial pattern
rs_pattern_clustered <- create_spatial_pattern(
  polygon = plgn,
  resolution = 0.1,
  spatial_pattern = "clustered",
  seed = 123)

ggplot() +
  geom_spatraster(data = rs_pattern_clustered) +
  scale_fill_continuous(type = "viridis") +
  theme_minimal()

# 3. User defined spatial pattern
# Large scale clustering
rs_pattern_large <- create_spatial_pattern(
  polygon = plgn,
  resolution = 0.1,
  spatial_pattern = 100,
  seed = 123)

ggplot() +
  geom_spatraster(data = rs_pattern_large) +
  scale_fill_continuous(type = "viridis") +
  theme_minimal()
```

`filter_observations` *Filter detected occurrences*

Description

This function filters observations from all occurrences based on the `sampling_status` column, typically created by the `sample_observations()` function.

Usage

```
filter_observations(observations_total, invert = FALSE)
```

Arguments

`observations_total`	An sf object with POINT geometry or a simple dataframe with `sampling_status` column containing values `"detected"`. This format is typically created by the `sample_observations()` function.
`invert`	Logical. If `FALSE` (default), the function filters to retain only `"detected"` occurrences. If `TRUE`, it filters out `"detected"` occurrences and retains all other occurrences.

Value

A data frame or an sf object with POINT geometry containing the filtered observations. If `invert = FALSE`, the function returns detected occurrences. If `invert = TRUE`, it returns all other occurrences.

See Also

Other main: `add_coordinate_uncertainty()`, `grid_designation()`, `sample_observations()`, `simulate_occurrences()`

Examples

```
# Create dataframe with sampling status column
occurrences_data <- data.frame(
    time_point = 1,
    sampling_prob = seq(0.5, 1, 0.1),
    sampling_status = rep(c("undetected", "detected"), each = 3)
  )

# Keep detected occurrences
filter_observations(occurrences_data)

# Keep undetected occurrences
filter_observations(occurrences_data, invert = TRUE)
```

`generate_taxonomy` *Generate a taxonomic hierarchy*

Description

This function generates a random taxonomic hierarchy for a specified numbers of species, genera, families, orders, classes, phyla, and kingdoms. The output is a data frame with the hierarchical classification for each species.

Usage

```
generate_taxonomy(
  num_species,
  num_genera,
  num_families,
  num_orders = 1,
  num_classes = 1,
  num_phyla = 1,
  num_kingdoms = 1,
  seed = NA
)
```

Arguments

`num_species`	Number of species to generate, or a dataframe. With a dataframe, the function will create a species with taxonomic hierarchy for each row. The original columns of the dataframe will be retained in the output.
`num_genera`	Number of genera to generate.
`num_families`	Number of families to generate.
`num_orders`	Number of orders to generate. Defaults to 1.
`num_classes`	Number of classes to generate. Defaults to 1.
`num_phyla`	Number of phyla to generate. Defaults to 1.
`num_kingdoms`	Number of kingdoms to generate. Defaults to 1.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Details

The function works by randomly assigning species to genera, genera to families, families to orders, orders to classes, classes to phyla, and phyla to kingdoms. Sampling is done with replacement, allowing multiple lower-level taxa (e.g., species) to be assigned to the same higher-level taxon (e.g., genus).

Value

A data frame with the taxonomic classification of each species. If `num_species` is a dataframe, the taxonomic classification is added to this input dataframe. The original columns of the dataframe will be retained in the output.

See Also

Other multispecies: `map_add_coordinate_uncertainty()`, `map_filter_observations()`, `map_grid_designation()`, `map_sample_observations()`, `map_simulate_occurrences()`

Examples

```
# 1. Create simple taxonomic hierarchy
generate_taxonomy(
  num_species = 5,
  num_genera = 3,
  num_families = 2,
  seed = 123)

# 2. Add taxonomic hierarchy to a dataframe
existing_df <- data.frame(
  count = c(1, 2, 5, 4, 8, 9, 3),
  det_prob = c(0.9, 0.9, 0.9, 0.8, 0.5, 0.2, 0.2)
  )

generate_taxonomy(
  num_species = existing_df,
  num_genera = 4,
  num_families = 2,
  seed = 125)
```

---

grid_designation *Observations to grid designation to create a data cube*

Description

This function designates observations to cells of a given grid to create an aggregated data cube.

Usage

```
grid_designation(
  observations,
  grid,
  id_col = "row_names",
  seed = NA,
  aggregate = TRUE,
  randomisation = c("uniform", "normal"),
  p_norm = ifelse(tolower(randomisation[1]) == "uniform", NA, 0.95)
)
```

Arguments

observations	An sf object with POINT geometry and a `time_point` and `coordinateUncertaintyInMeters` column. If the former column is not present, the function will assume a single time point. If the latter column is not present, the function will assume no uncertainty (zero meters) around the observation points.
grid	An sf object with POLYGON geometry (usually a grid) to which observations should be designated.

id_col	The column name containing unique IDs for each grid cell. If "row_names" (the default), a new column cell_code is created where the row names represent the unique IDs.
seed	A positive numeric value setting the seed for random number generation to ensure reproducibility. If NA (default), then set.seed() is not called at all. If not NA, then the random number generator state is reset (to the state before calling this function) upon exiting this function.
aggregate	Logical. If TRUE (default), returns data cube in aggregated form (grid with the number of observations per grid cell). Otherwise, returns sampled points within the uncertainty circle.
randomisation	Character. Method used for sampling within the uncertainty circle around each observation. "uniform" (default) means each point in the uncertainty circle has an equal probability of being selected. The other option is "normal", where a point is sampled from a bivariate Normal distribution with means equal to the observation point and variance such that p_norm % of all possible samples from this Normal distribution fall within the uncertainty circle. See sample_from_binormal_circle().
p_norm	A numeric value between 0 and 1, used only if randomisation = "normal". The proportion of all possible samples from a bivariate Normal distribution that fall within the uncertainty circle. Default is 0.95.

Value

If aggregate = TRUE, an sf object with POLYGON geometry containing the grid cells, an n column with the number of observations per grid cell, and a min_coord_uncertainty column with the minimum coordinate uncertainty per grid cell. If aggregate = FALSE, an sf object with POINT geometry containing the sampled observations within the uncertainty circles, and a coordinateUncertaintyInMeters column with the coordinate uncertainty for each observation.

See Also

Other main: add_coordinate_uncertainty(), filter_observations(), sample_observations(), simulate_occurrences()

Examples

```
library(sf)
library(dplyr)

# Create four random points
n_points <- 4
xlim <- c(3841000, 3842000)
ylim <- c(3110000, 3112000)
coordinate_uncertainty <- rgamma(n_points, shape = 5, rate = 0.1)

observations_sf <- data.frame(
  lat = runif(n_points, ylim[1], ylim[2]),
  long = runif(n_points, xlim[1], xlim[2]),
```

```
  time_point = 1,
  coordinateUncertaintyInMeters = coordinate_uncertainty
) %>%
  st_as_sf(coords = c("long", "lat"), crs = 3035)

# Add buffer uncertainty in meters around points
observations_buffered <- observations_sf %>%
  st_buffer(observations_sf$coordinateUncertaintyInMeters)

# Create grid
grid_df <- st_make_grid(
  observations_buffered,
  square = TRUE,
  cellsize = c(200, 200)
) %>%
  st_sf()

# Create occurrence cube
grid_designation(
  observations = observations_sf,
  grid = grid_df,
  seed = 123
)
```

`map_add_coordinate_uncertainty` *Map* `add_coordinate_uncertainty()` *over multiple species*

Description

This function executes `add_coordinate_uncertainty()` over multiple rows of a dataframe, representing different species, with potentially different function arguments over multiple columns.

Usage

```
map_add_coordinate_uncertainty(df, nested = TRUE, arg_list = NA)
```

Arguments

`df`	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping `add_coordinate_uncertainty` for each species. Columns not used by this function will be retained in the output.
`nested`	Logical. If `TRUE` (default), retains list-column containing sf objects calculated by `add_coordinate_uncertainty()`. Otherwise, expands this list-column into rows and columns.

arg_list — A named list or NA. If NA (default), the function assumes column names in df are identical to argument names of add_coordinate_uncertainty(). If column names differ, they must to be specified as a named list where the names are the argument names of add_coordinate_uncertainty(), and the associated values are the corresponding column names in df.

Value

In case of nested = TRUE, a dataframe identical to df, but each sf object with POINT geometry in the list-column observations now has an additional column coordinateUncertaintyInMeters added by add_coordinate_uncertainty(). In case of nested = FALSE, this list-column is expanded into additional rows and columns.

See Also

Other multispecies: generate_taxonomy(), map_filter_observations(), map_grid_designation(), map_sample_observations(), map_simulate_occurrences()

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 200),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  detection_probability = c(0.8, 0.9, 1),
  invert = FALSE,
  coords_uncertainty_meters = c(25, 30, 50),
  seed = 123)

# Simulate occurrences
sim_occ1 <- map_simulate_occurrences(df = species_dataset_df)

# Sample observations
samp_obs1 <- map_sample_observations(df = sim_occ1)

# Filter observations
filter_obs1 <- map_filter_observations(df = samp_obs1)

# Add coordinate uncertainty
```

```
obs_uncertainty_nested <- map_add_coordinate_uncertainty(df = filter_obs1)
obs_uncertainty_nested


## Example with deviating column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df2 <- species_dataset_df %>%
  rename(polygon = species_range,
         sd = sd_step,
         det_prob = detection_probability,
         inv = invert,
         coord_uncertainty = coords_uncertainty_meters)

# Create named list for argument conversion
arg_conv_list <- list(
    species_range = "polygon",
    sd_step = "sd",
    detection_probability = "det_prob",
    invert = "inv",
    coords_uncertainty_meters = "coord_uncertainty"
  )

# Simulate occurrences
sim_occ2 <- map_simulate_occurrences(
  df = species_dataset_df2,
  arg_list = arg_conv_list)

# Sample observations
samp_obs2 <- map_sample_observations(
  df = sim_occ2,
  arg_list = arg_conv_list)

# Filter observations
filter_obs2 <- map_filter_observations(
  df = samp_obs2,
  arg_list = arg_conv_list)

# Add coordinate uncertainty
map_add_coordinate_uncertainty(
  df = filter_obs2,
  arg_list = arg_conv_list)
```

`map_filter_observations` *Map* `filter_observations()` *over multiple species*

Description

This function executes `filter_observations()` over multiple rows of a dataframe, representing different species, with potentially different function arguments over multiple columns.

Usage

```
map_filter_observations(df, nested = TRUE, arg_list = NA)
```

Arguments

df	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping `filter_observations()` for each species. Columns not used by this function will be retained in the output.
nested	Logical. If `TRUE` (default), retains list-column containing sf objects/dataframes calculated by `filter_observations()`. Otherwise, expands this list-column into rows and columns.
arg_list	A named list or `NA`. If `NA` (default), the function assumes column names in `df` are identical to argument names of `filter_observations()`. If column names differ, they must be specified as a named list where the names are the argument names of `filter_observations()`, and the associated values are the corresponding column names in `df`.

Value

In case of `nested = TRUE`, a dataframe identical to `df`, with an extra list-column called `observations` containing an sf object with POINT geometry or simple dataframe for each row computed by `filter_observations()`. In case of `nested = FALSE`, this list-column is expanded into additional rows and columns.

See Also

Other multispecies: `generate_taxonomy()`, `map_add_coordinate_uncertainty()`, `map_grid_designation()`, `map_sample_observations()`, `map_simulate_occurrences()`

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 200),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  detection_probability = c(0.8, 0.9, 1),
```

```
  invert = FALSE,
  seed = 123)

# Simulate occurrences
sim_occ1 <- map_simulate_occurrences(df = species_dataset_df)

# Sample observations
samp_obs1 <- map_sample_observations(df = sim_occ1)

# Filter observations
filter_obs_nested <- map_filter_observations(df = samp_obs1)
filter_obs_nested

## Example with deviating column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df2 <- species_dataset_df %>%
  rename(polygon = species_range,
         sd = sd_step,
         det_prob = detection_probability,
         inv = invert)

# Create named list for argument conversion
arg_conv_list <- list(
    species_range = "polygon",
    sd_step = "sd",
    detection_probability = "det_prob",
    invert = "inv"
  )

# Simulate occurrences
sim_occ2 <- map_simulate_occurrences(
  df = species_dataset_df2,
  arg_list = arg_conv_list)

# Sample observations
samp_obs2 <- map_sample_observations(
  df = sim_occ2,
  arg_list = arg_conv_list)

# Filter observations
map_filter_observations(
  df = samp_obs2,
  arg_list = arg_conv_list)
```

`map_grid_designation` *Map* `grid_designation()` *over multiple species*

Description

This function executes `grid_designation()` over multiple rows of a dataframe, representing different species, with potentially different function arguments over multiple columns.

Usage

```
map_grid_designation(df, nested = TRUE, arg_list = NA)
```

Arguments

df	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping grid_designation() for each species. Columns not used by this function will be retained in the output.
nested	Logical. If TRUE (default), retains list-column containing sf objects calculated by grid_designation(). Otherwise, expands this list-column into rows and columns.
arg_list	A named list or NA. If NA (default), the function assumes column names in df are identical to argument names of grid_designation(). If column names differ, they must to be specified as a named list where the names are the argument names of grid_designation(), and the associated values are the corresponding column names in df.

Value

In case of nested = TRUE, a dataframe identical to df, but each sf object with POINT geometry in the list-column observations now has an additional column coordinateUncertaintyInMeters added by grid_designation(). In case of nested = FALSE, this list-column is expanded into additional rows and columns.

See Also

Other multispecies: generate_taxonomy(), map_add_coordinate_uncertainty(), map_filter_observations() map_sample_observations(), map_simulate_occurrences()

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

# Create grid
cube_grid <- st_make_grid(
    st_buffer(plgn, 25),
    n = c(20, 20),
    square = TRUE) %>%
  st_sf()

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
```

```
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 200),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  detection_probability = c(0.8, 0.9, 1),
  invert = FALSE,
  coords_uncertainty_meters = c(25, 30, 50),
  grid = rep(list(cube_grid), 3),
  seed = 123)

# Simulate occurrences
sim_occ1 <- map_simulate_occurrences(df = species_dataset_df)

# Sample observations
samp_obs1 <- map_sample_observations(df = sim_occ1)

# Filter observations
filter_obs1 <- map_filter_observations(df = samp_obs1)

# Add coordinate uncertainty
obs_uncertainty1 <- map_add_coordinate_uncertainty(df = filter_obs1)

# Grid designation
occ_cube_nested <- map_grid_designation(df = obs_uncertainty1)
occ_cube_nested

## Example with deviating column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df2 <- species_dataset_df %>%
  rename(polygon = species_range,
         sd = sd_step,
         det_prob = detection_probability,
         inv = invert,
         coord_uncertainty = coords_uncertainty_meters,
         raster = grid)

# Create named list for argument conversion
arg_conv_list <- list(
    species_range = "polygon",
    sd_step = "sd",
    detection_probability = "det_prob",
    invert = "inv",
    coords_uncertainty_meters = "coord_uncertainty",
    grid = "raster"
  )

# Simulate occurrences
sim_occ2 <- map_simulate_occurrences(
  df = species_dataset_df2,
  arg_list = arg_conv_list)
```

```
# Sample observations
samp_obs2 <- map_sample_observations(
  df = sim_occ2,
  arg_list = arg_conv_list)

# Filter observations
filter_obs2 <- map_filter_observations(
  df = samp_obs2,
  arg_list = arg_conv_list)

# Add coordinate uncertainty
obs_uncertainty2 <- map_add_coordinate_uncertainty(
  df = filter_obs2,
  arg_list = arg_conv_list)

# Grid designation
map_grid_designation(
  df = obs_uncertainty2,
  arg_list = arg_conv_list)
```

`map_sample_observations` *Map* `sample_observations()` *over multiple species*

Description

This function executes `sample_observations()` over multiple rows of a dataframe, representing different species, with potentially different function arguments over multiple columns.

Usage

```
map_sample_observations(df, nested = TRUE, arg_list = NA)
```

Arguments

`df`	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping `sample_observations()` for each species. Columns not used by this function will be retained in the output.
`nested`	Logical. If `TRUE` (default), retains list-column containing sf objects calculated by `sample_observations()`. Otherwise, expands this list-column into rows and columns.
`arg_list`	A named list or `NA`. If `NA` (default), the function assumes column names in `df` are identical to argument names of `sample_observations()`. If column names differ, they must be specified as a named list where the names are the argument names of `sample_observations()`, and the associated values are the corresponding column names in `df`.

Value

In case of `nested = TRUE`, a dataframe identical to `df`, with an extra list-column called `occurrences` containing an sf object with POINT geometry for each row computed by `sample_observations()`. In case of `nested = FALSE`, this list-column is expanded into additional rows and columns.

See Also

Other multispecies: `generate_taxonomy()`, `map_add_coordinate_uncertainty()`, `map_filter_observations()`, `map_grid_designation()`, `map_simulate_occurrences()`

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 200),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  detection_probability = c(0.8, 0.9, 1),
  seed = 123)

# Simulate occurrences
sim_occ1 <- map_simulate_occurrences(df = species_dataset_df)

# Sample observations
samp_obs_nested <- map_sample_observations(df = sim_occ1)
samp_obs_nested

## Example with deviating column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df2 <- species_dataset_df %>%
  rename(polygon = species_range,
         sd = sd_step,
         det_prob = detection_probability)

# Create named list for argument conversion
arg_conv_list <- list(
    species_range = "polygon",
    sd_step = "sd",
    detection_probability = "det_prob"
```

```
  )

# Simulate occurrences
sim_occ2 <- map_simulate_occurrences(
  df = species_dataset_df2,
  arg_list = arg_conv_list)

# Sample observations
map_sample_observations(
  df = sim_occ2,
  arg_list = arg_conv_list)
```

`map_simulate_occurrences` *Map* `simulate_occurrences()` *over multiple species*

Description

This function executes `simulate_occurrences()` over multiple rows of a dataframe, representing different species, with potentially different function arguments over multiple columns.

Usage

```
map_simulate_occurrences(df, nested = TRUE, arg_list = NA)
```

Arguments

`df`	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping `simulate_occurrences()` for each species. Columns not used by this function will be retained in the output.
`nested`	Logical. If `TRUE` (default), retains list-column containing sf objects calculated by `simulate_occurrences()`. Otherwise, expands this list-column into rows and columns.
`arg_list`	A named list or `NA`. If `NA` (default), the function assumes column names in `df` are identical to argument names of `simulate_occurrences()` and the function specified in its `temporal_function` argument. If column names differ, they must be specified as a named list where the names are the argument names of `simulate_occurrences()` or the function specified in its `temporal_function` argument, and the associated values are the corresponding column names in `df`.

Value

In case of `nested = TRUE`, a dataframe identical to `df`, with an extra list-column called `occurrences` containing an sf object with POINT geometry for each row computed by `simulate_occurrences()`. In case of `nested = FALSE`, this list-column is expanded into additional rows and columns.

See Also

Other multispecies: generate_taxonomy(), map_add_coordinate_uncertainty(), map_filter_observations() map_grid_designation(), map_sample_observations()

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 200),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  seed = 123)

# Simulate occurrences
sim_occ_nested <- map_simulate_occurrences(df = species_dataset_df)
sim_occ_nested

## Example with deviating column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df2 <- species_dataset_df %>%
  rename(polygon = species_range,
         sd = sd_step)

# Create named list for argument conversion
arg_conv_list <- list(
    species_range = "polygon",
    sd_step = "sd"
  )

# Simulate occurrences
map_simulate_occurrences(
  df = species_dataset_df2,
  arg_list = arg_conv_list)
```

`map_simulation_functions`

Map a cube simulation function over multiple rows of a dataframe

---

Description

This function executes a cube simulation function (`simulate_occurrences()`, `sample_observations()`, `filter_observations()`, `add_coordinate_uncertainty()`, or `grid_designation()`) over multiple rows of a dataframe with potentially different function arguments over multiple columns.

Usage

```
map_simulation_functions(f, df, nested = TRUE)
```

Arguments

`f`	One of five cube simulation functions: `simulate_occurrences()`, `sample_observations()`, `filter_observations()`, `add_coordinate_uncertainty()`, or `grid_designation()`.
`df`	A dataframe containing multiple rows, each representing a different species. The columns are function arguments with values used for mapping `f` for each species. Columns not used by this function will be retained in the output.
`nested`	Logical. If `TRUE` (default), retains list-column containing dataframes calculated by `f`. Otherwise, expands this list-column into rows and columns.

Value

In case of `nested = TRUE`, a dataframe identical to `df`, with an extra list-column called `mapped_col` containing an sf object for each row computed by the function specified in `f`. In case of `nested = FALSE`, this list-column is expanded into additional rows and columns.

Examples

```
# Load packages
library(sf)
library(dplyr)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Example with simple column names
# Specify dataframe for 3 species with custom function arguments
species_dataset_df <- tibble(
  taxonID = c("species1", "species2", "species3"),
  species_range = rep(list(plgn), 3),
  initial_average_occurrences = c(50, 100, 500),
  n_time_points = rep(6, 3),
  temporal_function = c(simulate_random_walk, simulate_random_walk, NA),
  sd_step = c(1, 1, NA),
  spatial_pattern = "random",
  seed = 123)

# Simulate occurrences
sim_occ_raw <- map_simulation_functions(
```

```
  f = simulate_occurrences,
  df = species_dataset_df)
sim_occ_raw

# Unnest output and create sf object
sim_occ_raw_unnested <- map_simulation_functions(
  f = simulate_occurrences,
  df = species_dataset_df,
  nested = FALSE)

sim_occ_raw_unnested %>%
   st_sf()
```

`sample_from_binormal_circle` *Sample from a circle using the bivariate Normal distribution*

Description

This function samples a new observations point of a species within the uncertainty circle around each observation assuming a bivariate Normal distribution.

Usage

```
sample_from_binormal_circle(observations, p_norm = 0.95, seed = NA)
```

Arguments

`observations`	An sf object with POINT geometry and a `time_point` and `coordinateUncertaintyInMeters` column. If the former column is not present, the function will assume a single time point. If the latter column is not present, the function will assume no uncertainty (zero meters) around the observation points.
`p_norm`	A numeric value between 0 and 1. The proportion of all possible samples from a bivariate Normal distribution that fall within the uncertainty circle. Default is 0.95. See Details.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Details

A new observation point is sampled from a bivariate Normal distribution with means equal to the X and Y coordinates of its original observation point and variances equal to (-`coordinateUncertaintyInMeters`^2) / (2 * log(1 - `p_norm`)), ensuring `p_norm` % of all possible samples fall within the uncertainty circle.

Value

An sf object with POINT geometry containing the locations of the sampled occurrences and a `coordinateUncertaintyInMeters` column containing the coordinate uncertainty for each observation.

See Also

Other designation: `sample_from_uniform_circle()`

Examples

```
library(sf)
library(dplyr)

# Create four random points
n_points <- 4
xlim <- c(3841000, 3842000)
ylim <- c(3110000, 3112000)
coordinate_uncertainty <- rgamma(n_points, shape = 5, rate = 0.1)

observations_sf <- data.frame(
  lat = runif(n_points, ylim[1], ylim[2]),
  long = runif(n_points, xlim[1], xlim[2]),
  time_point = 1,
  coordinateUncertaintyInMeters = coordinate_uncertainty
) %>%
  st_as_sf(coords = c("long", "lat"), crs = 3035)

# Sample points within uncertainty circles according to normal rules
sample_from_binormal_circle(
  observations = observations_sf,
  p_norm = 0.95,
  seed = 123
)
```

`sample_from_uniform_circle` *Sample from a circle using the Uniform distribution*

Description

This function samples a new observations point of a species within the uncertainty circle around each observation assuming a Uniform distribution.

Usage

```
sample_from_uniform_circle(observations, seed = NA)
```

Arguments

`observations`	An sf object with POINT geometry and a `time_point` and `coordinateUncertaintyInMeters` column. If the former column is not present, the function will assume a single time point. If the latter column is not present, the function will assume no uncertainty (zero meters) around the observation points.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

An sf object with POINT geometry containing the locations of the sampled occurrences and a `coordinateUncertaintyInMeters` column containing the coordinate uncertainty for each observation.

See Also

Other designation: `sample_from_binormal_circle()`

Examples

```
library(sf)

# Create four random points
n_points <- 4
xlim <- c(3841000, 3842000)
ylim <- c(3110000, 3112000)
coordinate_uncertainty <- rgamma(n_points, shape = 5, rate = 0.1)

observations_sf <- data.frame(
  lat = runif(n_points, ylim[1], ylim[2]),
  long = runif(n_points, xlim[1], xlim[2]),
  time_point = 1,
  coordinateUncertaintyInMeters = coordinate_uncertainty
) %>%
  st_as_sf(coords = c("long", "lat"), crs = 3035)

# Sample points within uncertainty circles according to uniform rules
sample_from_uniform_circle(
  observations = observations_sf,
  seed = 123
)
```

`sample_observations` *Sample observations from a larger occurrence dataset*

Description

The function computes observations from occurrences based on detection probability and sampling bias by implementing a Bernoulli trial.

Usage

```
sample_observations(
  occurrences,
  detection_probability = 1,
  sampling_bias = c("no_bias", "polygon", "manual"),
  bias_area = NA,
  bias_strength = 1,
  bias_weights = NA,
  seed = NA
)
```

Arguments

`occurrences`	An sf object with POINT geometry representing the occurrences.
`detection_probability`	A numeric value between 0 and 1 representing the probability of detecting the species.
`sampling_bias`	A character string specifying the method to generate a sampling bias. Options are `"no_bias"`, `"polygon"`, or `"manual"`. `"no_bias"` No bias is applied (default). `"polygon"` Bias the sampling within a polygon. Provide the polygon to `bias_area` and the bias strength to `bias_strength`. `"manual"` Bias the sampling manually using a grid. Provide the grid layer in which each cell contains the probability of being sampled to `bias_weights`.
`bias_area`	An `sf` object with POLYGON geometry, or `NA`. Only used if `sampling_bias = "polygon"`. This defines the area in which the sampling will be biased.
`bias_strength`	A positive numeric value, or `NA`. Only used if `sampling_bias = "polygon"`. The value represents the strength of the bias to be applied within the `bias_area`. Values greater than 1 will increase the sampling probability within the polygon relative to outside (oversampling), while values between 0 and 1 will decrease it (undersampling). For instance, a value of 50 will make the probability 50 times higher within the `bias_area` compared to outside, whereas a value of 0.5 will make it half as likely.
`bias_weights`	A grid layer (an sf object with POLYGON geometry), or `NA`. Only used if `sampling_bias = "manual"`. The grid of bias weights to be applied. This sf object should contain a `bias_weight` column with the weights per grid cell. Higher weights increase the probability of sampling. Weights can be numeric values between 0 and 1 or positive integers, which will be rescaled to values between 0 and 1.

`seed` A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

An sf object with POINT geometry containing the locations of the occurrence with detection status. The object includes the following columns:

`detection_probability` The detection probability for each occurrence (will be the same for all).

`bias_weight` The sampling probability based on sampling bias for each occurrence.

`sampling_probability` The combined sampling probability from detection probability and sampling bias for each occurrence.

`sampling_status` Indicates whether the occurrence was detected (`"detected"`) or not (`"undetected"`). Detected occurrences are called observations.

See Also

Other main: `add_coordinate_uncertainty()`, `filter_observations()`, `grid_designation()`, `simulate_occurrences()`

Examples

```
# Load packages
library(sf)
library(dplyr)

# Simulate some occurrence data with coordinates and time points
num_points <- 10
occurrences <- data.frame(
  lon = runif(num_points, min = -180, max = 180),
  lat = runif(num_points, min = -90, max = 90),
  time_point = 0
  )

# Convert the occurrence data to an sf object
occurrences_sf <- st_as_sf(occurrences, coords = c("lon", "lat"))

# 1. Sample observations without sampling bias
sample_observations(
  occurrences_sf,
  detection_probability = 0.8,
  sampling_bias = "no_bias",
  seed = 123
  )

# 2. Sample observations with sampling bias in a polygon
# Create bias_area polygon overlapping two of the points
selected_observations <- st_union(occurrences_sf[2:3,])
```

```
bias_area <- st_convex_hull(selected_observations) %>%
  st_buffer(dist = 50) %>%
  st_as_sf()

sample_observations(
  occurrences_sf,
  detection_probability = 0.8,
  sampling_bias = "polygon",
  bias_area = bias_area,
  bias_strength = 2,
  seed = 123
  )

# 3. Sample observations with sampling bias given manually in a grid
# Create raster grid with bias weights between 0 and 1
grid <- st_make_grid(occurrences_sf) %>%
  st_sf() %>%
  mutate(bias_weight = runif(n(), min = 0, max = 1))

sample_observations(
  occurrences_sf,
  detection_probability = 0.8,
  sampling_bias = "manual",
  bias_weights = grid,
  seed = 123
  )
```

`sample_occurrences_from_raster` *Sample occurrences from spatial random field*

Description

This function draws point occurrences from a spatial random field represented by a raster. Points are sampled based on the values in the raster, with the number of occurrences specified for each time step.

Usage

```
sample_occurrences_from_raster(raster, time_series, seed = NA)
```

Arguments

`raster`	A SpatRaster object (see `terra::rast()`).
`time_series`	A vector with the number of occurrences per time point.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

An sf object with POINT geometry containing the locations of the simulated occurrences, a `time_point` column indicating the associated time point for each occurrence and columns used as weights for sampling. If the raster is created with `create_spatial_pattern()`, the column `sampling_p1` is used.

See Also

Other occurrence: `create_spatial_pattern()`, `simulate_random_walk()`, `simulate_timeseries()`

Examples

```
# Load packages
library(sf)
library(ggplot2)
library(tidyterra)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

## Medium scale clustering
# Create the random field
rs_pattern_clustered <- create_spatial_pattern(
  polygon = plgn,
  resolution = 0.1,
  spatial_pattern = "clustered",
  seed = 123)

# Sample 200 occurrences from random field
pts_occ_clustered <- sample_occurrences_from_raster(
  raster = rs_pattern_clustered,
  time_series = 200,
  seed = 123)

ggplot() +
  geom_spatraster(data = rs_pattern_clustered) +
  geom_sf(data = pts_occ_clustered) +
  scale_fill_continuous(type = "viridis") +
  theme_minimal()

## Large scale clustering
# Create the random field
rs_pattern_large <- create_spatial_pattern(
  polygon = plgn,
  resolution = 0.1,
  spatial_pattern = 100,
  seed = 123)

# Sample 200 occurrences from random field
pts_occ_large <- sample_occurrences_from_raster(
  raster = rs_pattern_large,
  time_series = 200,
```

```
  seed = 123)

ggplot() +
  geom_spatraster(data = rs_pattern_large) +
  geom_sf(data = pts_occ_large) +
  scale_fill_continuous(type = "viridis") +
  theme_minimal()
```

`simulate_occurrences` *Simulate species occurrences within a spatiotemporal scope*

Description

This function simulates occurrences of a species within a specified spatial and/or temporal extent.

Usage

```
simulate_occurrences(
  species_range,
  initial_average_occurrences = 50,
  spatial_pattern = c("random", "clustered"),
  n_time_points = 1,
  temporal_function = NA,
  ...,
  seed = NA
)
```

Arguments

`species_range`	An sf object with POLYGON geometry indicating the spatial extend to simulate occurrences.
`initial_average_occurrences`	A positive numeric value indicating the average number of occurrences to be simulated within the extent of `species_range` at the first time point. This value serves as the mean (lambda) of a Poisson distribution.
`spatial_pattern`	Specifies the spatial pattern of occurrences. It can be a character string (`"random"` or `"clustered"`) or a numeric value 1 (1 means random distribution, larger values indicate more clustering). The default is `"random"`. `"clustered"` corresponds to a value of 10. See `create_spatial_pattern()`.
`n_time_points`	A positive integer specifying the number of time points to simulate.
`temporal_function`	A function generating a trend in number of occurrences over time, or `NA` (default). If `n_time_points` > 1 and a function is provided, it defines the temporal pattern of number of occurrences.
`...`	Additional arguments to be passed to `temporal_function`.

`seed` A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

An sf object with POINT geometry containing the locations of the simulated occurrences, a `time_point` column indicating the associated time point for each occurrence and a `sampling_p1` column indicating the sampling probability associated with the spatial pattern (see `create_spatial_pattern()`).

See Also

Other main: `add_coordinate_uncertainty()`, `filter_observations()`, `grid_designation()`, `sample_observations()`

Examples

```
# Load packages
library(sf)
library(ggplot2)

# Create polygon
plgn <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7, 9, 5, 2))))

# 1. Random spatial pattern with 4 time points
occ_sf <- simulate_occurrences(
  species_range = plgn,
  n_time_points = 4,
  initial_average_occurrences = 100,
  seed = 123)

ggplot() +
 geom_sf(data = occ_sf) +
 geom_sf(data = plgn, fill = NA) +
 facet_wrap("time_point") +
 labs(
      title = "Occurrences with random spatial and temporal pattern",
      subtitle = "4 time steps") +
 theme_minimal()

# 2. Highly clustered spatial pattern with 6 time points
occ_sf_100 <- simulate_occurrences(
  species_range = plgn,
  spatial_pattern = 100,
  n_time_points = 6,
  initial_average_occurrences = 100,
  seed = 123)

ggplot() +
  geom_sf(data = occ_sf_100) +
```

```
  geom_sf(data = plgn, fill = NA) +
  facet_wrap("time_point") +
  labs(
       title = "Occurrences with structured spatial and temporal pattern",
       subtitle = "6 time steps") +
  theme_minimal()
```

`simulate_random_walk` *Simulate a random walk over time*

Description

This function simulates a timeseries for the average number of occurrences of a species using a random walk over time.

Usage

```
simulate_random_walk(
  initial_average_occurrences = 50,
  n_time_points = 10,
  sd_step = 0.05,
  seed = NA
)
```

Arguments

`initial_average_occurrences`	A positive numeric value indicating the average number of occurrences to be simulated at the first time point.
`n_time_points`	A positive integer specifying the number of time points to simulate.
`sd_step`	A positive numeric value indicating the standard deviation of the random steps.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

A vector of integers of length `n_time_points` with the average number of occurrences.

See Also

Other occurrence: `create_spatial_pattern()`, `sample_occurrences_from_raster()`, `simulate_timeseries()`

Examples

```
simulate_random_walk(
  initial_average_occurrences = 50,
  n_time_points = 10,
  sd_step = 1,
  seed = 123
)
```

---

simulate_timeseries *Simulate timeseries for species occurrences*

Description

This function simulates a timeseries for the number of occurrences of a species.

Usage

```
simulate_timeseries(
  initial_average_occurrences = 50,
  n_time_points = 1,
  temporal_function = NA,
  ...,
  seed = NA
)
```

Arguments

`initial_average_occurrences`	A positive numeric value indicating the average number of occurrences to be simulated at the first time point. This value serves as the mean (lambda) of a Poisson distribution.
`n_time_points`	A positive integer specifying the number of time points to simulate.
`temporal_function`	A function generating a trend in number of occurrences over time, or `NA` (default). If `n_time_points` > 1 and a function is provided, it defines the temporal pattern of number of occurrences.
`...`	Additional arguments to be passed to `temporal_function`.
`seed`	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.

Value

A vector of integers of length `n_time_points` with the number of occurrences.

See Also

Other occurrence: create_spatial_pattern(), sample_occurrences_from_raster(), simulate_random_walk(

Examples

```
# 1. Use the function simulate_random_walk()
simulate_timeseries(
  initial_average_occurrences = 50,
  n_time_points = 10,
  temporal_function = simulate_random_walk,
  sd_step = 1,
  seed = 123
)

# 2. Using your own custom function, e.g. this linear function
my_own_linear_function <- function(
    initial_average_occurrences = initial_average_occurrences,
    n_time_points = n_time_points,
    coef) {
  # Calculate new average abundances over time
  time <- seq_len(n_time_points) - 1
  lambdas <- initial_average_occurrences + (coef * time)

  # Identify where the lambda values become 0 or lower
  zero_or_lower_index <- which(lambdas <= 0)

  # If any lambda becomes 0 or lower, set all subsequent lambdas to 0
  if (length(zero_or_lower_index) > 0) {
    zero_or_lower_indices <- zero_or_lower_index[1]:n_time_points
    lambdas[zero_or_lower_indices] <- 0
  }

  # Return average abundances
  return(lambdas)
}

# Draw n_sim number of occurrences from Poisson distribution using
# the custom function
n_sim <- 10
n_time_points <- 50
slope <- 1
list_abundances <- vector("list", length = n_sim)

# Loop n_sim times over simulate_timeseries()
for (i in seq_len(n_sim)) {
  abundances <- simulate_timeseries(
    initial_average_occurrences = 50,
    n_time_points = n_time_points,
    temporal_function = my_own_linear_function,
    coef = slope
  )
```

```
  list_abundances[[i]] <- data.frame(
    time = seq_along(abundances),
    abundance = abundances,
    sim = i
  )
}

# Combine list of dataframes
data_abundances <- do.call(rbind.data.frame, list_abundances)

# Plot the simulated abundances over time using ggplot2
library(ggplot2)
ggplot(data_abundances, aes(x = time, y = abundance, colour = factor(sim))) +
  geom_line() +
  labs(
    x = "Time", y = "Species abundance",
    title = paste(
      n_sim, "simulated trends using custom linear function",
      "with slope", slope
    )
  ) +
  scale_y_continuous(limits = c(0, NA)) +
  scale_x_continuous(breaks = seq(0, n_time_points, 5)) +
  theme_minimal() +
  theme(legend.position = "")
```

Index