Package ‘shar’

November 15, 2019

Type Package
Title Species-Habitat Associations
Version 1.1
Maintainer Maximillian H.K. Hesselbarth <maximilian.hesselbarth@uni-goettingen.de>
Description Analyse species-habitat associations in R. Therefore, information about the
location of the species is needed and about the environmental conditions. To test
for significance habitat associations, one of the two components is randomized.
Methods are mainly based on Plotkin et al. (2000) <doi:10.1006/jtbi.2000.2158> and
License GPL-3
URL https://r-spatialecology.github.io/shar
BugReports https://github.com/r-spatialecology/shar/issues
Depends R (>= 3.1)
Imports classInt, graphics, grDevices, methods, raster, spatstat,
            stats, utils, Rcpp
RoxygenNote 6.1.1
Suggests covr, dplyr, testthat (>= 2.1.0), knitr, rmarkdown
VignetteBuilder knitr
Encoding UTF-8
LazyData true
LinkingTo Rcpp
NeedsCompilation yes
Author Maximillian H.K. Hesselbarth [aut, cre]
            (<https://orcid.org/0000-0003-1125-9918>),
            Marco Sciaini [aut] (<https://orcid.org/0000-0002-3042-5435>)
Repository CRAN
Date/Publication 2019-11-15 15:20:02 UTC
R topics documented:

- calculate_energy
- classify_habitats
- create_neighbourhood
- estimate_pcf_fast
- extract_points
- fit_point_process
- gamma_test
- landscape
- plot_energy
- plot_randomized_pattern
- plot_randomized_raster
- print.rd_mar
- print.rd_pat
- print.rd_ras
- randomize_raster
- random_walk
- rcpp_sample
- reconstruction
- reconstruct_pattern_cluster
- reconstruct_pattern_hetero
- reconstruct_pattern_homo
- reconstruct_pattern_marks
- results_habitat_association
- shar
- species_a
- species_b
- torus_trans
- translate_raster

Index

- calculate_energy

Description

Calculate mean energy

Usage

```r
calculate_energy(pattern, weights = c(0.5, 0.5), return_mean = FALSE,
                 comp_fast = 1000, verbose = TRUE)
```
calculate_energy

Arguments

- **pattern**: List with reconstructed patterns.
- **weights**: Weights used to calculate energy. The first number refers to Gest(r), the second number to pcf(r).
- **return_mean**: Return the mean energy.
- **comp_fast**: If pattern contains more points than threshold, summary functions are estimated in a computational fast way.
- **verbose**: Print progress report.

Details

The function calculates the mean energy (or deviation) between the observed pattern and all reconstructed patterns (for more information see Tscheschel & Stoyan (2006) or Wiegand & Moloney (2014)). The pair correlation function and the nearest neighbour distance function are used to describe the patterns. For large patterns `comp_fast = TRUE` decreases the computational demand because no edge correction is used and the pair correlation function is estimated based on Ripley’s K-function. For more information see `estimate_pcf_fast`.

Value

numeric

References


See Also

- `plot_energy`
- `reconstruct_pattern_homo`
- `reconstruct_pattern_hetero`
- `reconstruct_pattern_cluster`
- `plot_randomized_pattern`

Examples

```r
pattern_random <- fit_point_process(species_a, n_random = 19)
calculate_energy(pattern_random)
calculate_energy(pattern_random, return_mean = TRUE)

## Not run:
marks_sub <- spatstat::subset.ppp(species_a, select = dbh)
marks_recon <- reconstruct_pattern_marks(pattern_random$randomized[1], marks_sub, n_random = 19, max_runs = 1000)
calculate_energy(marks_recon, return_mean = FALSE)
```
## classify_habitats

### Description

Classify habitats

### Usage

```r
classify_habitats(raster, classes = 5, style = "fisher")
```

### Arguments

- `raster`: RasterLayer.
- `classes`: Number of classes.
- `style`: Style of classification.

### Details

Classifies a RasterLayer with continuous values into n discrete classes. Consequently, classes are non-overlapping (and left-closed). For more information see `classIntervals`.

### Value

RasterLayer

### References


### See Also

`classIntervals`
create_neighbourhood

Examples

landscape_classified <- classify_habitats(landscape, classes = 5)

create_neighbourhood create_neighbourhood

description

Create neighbourough

Usage

create_neighbourhood(cells, matrix, directions = 4)

Arguments

cells  
matrix with cell ids of focal cells.

matrix  
matrix in which cells are located.

directions  
Cells neighbour rule: 4 (rook’s case), 8 (queen’s case).

Details

Get cell ids of all neighbouring cells. The neighbourough rule can be specified and is either rook’s case (4 neighbours) or queen’s case (8 neighbours).

Value

matrix

See Also

randomize_raster

Examples

mat <- matrix(1, nrow= 10, ncol = 10)
cell_id <- rbind(cbind(3,5), cbind(7,1))
create_neighbourhood(cell_id, mat)
estimate_pcf_fast

Description

Fast estimation of the pair correlation function

Usage

estimate_pcf_fast(pattern, ...)

Arguments

pattern Point pattern.
...
Arguments passed down to ‘Kest’ or ‘pcf.fv’.

Details

The functions estimates the pair correlation functions based on an estimation of Ripley’s K-function. This makes it computationally faster than estimating the pair correlation function directly. It is a wrapper around ‘Kest’ and ‘pcf.fv’.

Value

fv.object

References


See Also

Kest
pcf.fv

Examples

pcf_species_b <- estimate_pcf_fast(species_a)
**extract_points**

**Description**

Extract points

**Usage**

`extract_points(raster, pattern)`

**Arguments**

- `raster` : RasterLayer.
- `pattern` : Point pattern.

**Details**

The function extracts the number of points within each habitat.

**Value**

data.frame

**Examples**

```
landscape_classified <- classify_habitats(landscape, classes = 5)
extract_points(raster = landscape_classified, pattern = species_b)
```

**fit_point_process**

**Description**

Create random patterns by point process fitting

**Usage**

`fit_point_process(pattern, n_random = 1, process = "poisson", return_input = TRUE, simplify = FALSE, verbose = TRUE)`
Arguments

- **pattern**: List with reconstructed patterns.
- **n_random**: Number of randomized RasterLayers.
- **process**: What point process to use. Either 'poisson' or 'cluster'.
- **return_input**: The original input data is returned as last list entry.
- **simplify**: If n_random = 1 and return_input = FALSE only pattern will be returned.
- **verbose**: Print progress report.

Details

The function randomizes the observed pattern by fitting a point process to the data. It is possible to choose between a Poisson process or a Thomas cluster process.

Value

- list

References


Examples

```r
pattern_fitted <- fit_point_process(pattern = species_a, n_random = 39)
```

---

**gamma_test**

*Gamma test*

Description

Randomized data for species b using the gamma test.

Usage

```r
gamma_test
```

Format

- `rd_pat` object.
Example landscape (random cluster neutral landscape model).

Description

An example map to show landscapetools functionality generated with the 'nlm_fbm()' algorithm.

Usage

landscape

Format

A raster layer object.

Source

Simulated neutral landscape model with R. https://github.com/ropensci/NLMR/

plot_energy

Description

Plot energy of pattern reconstruction

Usage

plot_energy(pattern, col = NULL)

Arguments

pattern List with reconstructed patterns.
col Vector with colors. Must be as long as n_random.

Details

The function plots the decrease of the energy over time, i.e. the iterations. This can help to identify if enough max_runs where chosen for the reconstruction.

See Also

calculate_energy
reconstruct_pattern_homo
reconstruct_pattern_hetero
reconstruct_pattern_cluster
plot_randomized_pattern
Examples

```r
## Not run:
pattern_recon <- reconstruct_pattern_homo(species_a, n_random = 3, max_runs = 1000)
plot_energy(pattern_recon)

marks_sub <- spatstat::subset.ppp(species_a, select = dbh)
marks_recon <- reconstruct_pattern_marks(pattern_recon$randomized[[1]], marks_sub,
n_random = 1, max_runs = 1000)
plot_energy(marks_recon)

## End(Not run)
```

Description

Plot randomized pattern

Usage

```r
plot_randomized_pattern(pattern, what = "sf", probs = c(0.025, 0.975),
comp_fast = 1000, ask = TRUE, verbose = TRUE)
```

Arguments

- `pattern`: List with reconstructed patterns.
- `what`: Plot summary functions of point patterns (what = "sf") or acutal patterns (what = "pp").
- `probs`: Quantiles of randomized data used for envelope construction.
- `comp_fast`: If pattern contains more points than threshold, summary functions are estimated in a computational fast way.
- `ask`: If TRUE the user is asked to press <RETURN> before second summary function is plotted (only has influence if what = "sf" and method = "spatial").
- `verbose`: Print progress report.

Details

The function plots the pair correlation function and the nearest neighbour function the observed pattern and the reconstructed patterns (as "simulation envelopes"). For large patterns `comp_fast = TRUE` decreases the computational demand because no edge correction is used and the pair correlation function is estimated based on Ripley’s K-function. For more information see `estimate_pcf_fast`. It is also possible to plot 3 randomized patterns and the observed pattern using `what = "pp"`. 
Examples

```r
pattern_random <- fit_point_process(species_a, n_random = 19, process = "cluster")
plot_randomized_pattern(pattern_random)

plot_randomized_pattern(pattern_random, what = "pp")

## Not run:
marks_sub <- spatstat::subset.ppp(species_a, select = dbh)
marks_recon <- reconstruct_pattern_marks(pattern_random$randomized[[1]], marks_sub, 
n_random = 19, max_runs = 1000)
plot_randomized_pattern(marks_recon)

## End(Not run)
```

Description

Plot randomized raster

Usage

```r
plot_randomized_raster(raster, n = NULL, col, verbose = TRUE, nrow, ncol)
```

Arguments

- **raster**: List with randomized raster
- **n**: Number of randomized rasters to plot. See details for more information.
- **col**: Color palette used for plotting.
- **verbose**: Print messages.
- **nrow, ncol**: Number of rows and columns.

Details

Function to plot randomized rasters. If `n` is only a single number, `n` randomized rasters will be sampled. If `n` is a vector, the corresponding rasters will be plotted.

Value

- `plot`
Examples

```r
## Not run:
landscape_classes <- classify_habitats(raster = landscape, classes = 5)
landscape_random <- randomize_raster(raster = landscape_classes, n_random = 19)
plot_randomized_raster(landscape_random)
palette <- viridis::viridis(n = 5)
plot_randomized_raster(landscape_random, n = 5, col = palette, nrow = 3, ncol = 2)
## End(Not run)
```

Description

Print method for rd_mar object

Usage

```r
## S3 method for class 'rd_mar'
print(x, digits = 4, ...)
```

Arguments

- `x` Random patterns.
- `digits` Number of decimal places (round).
- `...` Arguments passed to `cat`

Details

Printing method for random patterns created with `reconstruct_pattern_marks`.

See Also

`reconstruct_pattern_marks`

Examples

```r
## Not run:
pattern_recon <- reconstruct_pattern_homo(species_a, n_random = 1, max_runs = 1000,
simplify = TRUE, return_input = FALSE)
marks_sub <- spatstat::subset.ppp(species_a, select = dbh)
marks_recon <- reconstruct_pattern_marks(pattern_recon, marks_sub, n_random = 19, max_runs = 1000)
print(marks_recon)
## End(Not run)
```
Description

Print method for rd_pat object

Usage

```r
## S3 method for class 'rd_pat'
print(x, digits = 4, ...)
```

Arguments

- `x`: Random patterns.
- `digits`: Number of decimal places (round).
- `...`: Arguments passed to `cat`

Details

Printing method for random patterns created with `reconstruct_pattern_homo`, `reconstruct_pattern_hetero`, `reconstruct_pattern_cluster` or `fit_point_process`.

See Also

- `reconstruct_pattern_homo`
- `reconstruct_pattern_hetero`
- `reconstruct_pattern_cluster`
- `fit_point_process`

Examples

```r
pattern_random <- fit_point_process(species_a, n_random = 199)
print(pattern_random)

# Not run:
pattern_recon <- reconstruct_pattern_hetero(species_b, n_random = 19, max_runs = 1000)
print(pattern_recon)

# End(Not run)
```
Description

Print method for rd_ras object

Usage

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

Arguments

x Random patterns.
...
Arguments passed to cat

Details

Printing method for random patterns created with randomize_raster.

See Also

randomize_raster

Examples

## Not run:
landscape_classified <- classify_habitats(landscape, classes = 5)
landscape_random <- randomize_raster(landscape_classified, n_random = 19)

print(landscape_random)

## End(Not run)

randomize_raster

Description

Randomization algorithm

Usage

randomize_raster(raster, n_random = 1, directions = 4,
return_input = TRUE, simplify = FALSE, verbose = TRUE)
**randomize_raster**

**Arguments**

- **raster** RasterLayer.
- **n_random** Number of randomizations.
- **directions** Cells neighbour rule: 4 (rook’s case), 8 (queen’s case).
- **return_input** The original input data is returned as last list entry
- **simplify** If n_random = 1 and return_input = FALSE only raster will be returned.
- **verbose** Print progress report.

**Details**

The function randomizes a habitat map (as RasterLayer) as proposed by Harms et al. (2001) as “randomized-habitats procedure”. The algorithm starts with an empty habitat map starts to assign random neighbouring cells to each habitat (in increasing order of abundance in observed map). We modified the procedure slightly by increasing a probability to jump to a non-neighbouring cell as the current patch becomes larger.

**Value**

list

**References**


**See Also**

- translate_raster
  - adjacent

**Examples**

```r
## Not run:
landscape_classified <- classify_habitats(landscape, classes = 5)
landscape_random <- randomize_raster(landscape_classified, n_random = 19)

## End(Not run)
```
random_walk  Random walk

**Description**
Randomization of the landscape data using the habitat randomization algorithm.

**Usage**
random_walk

**Format**
rd_ras object.

rcpp_sample  rcpp_sample

**Description**  Rcpp sample function

**Usage**
rcpp_sample(x, n, replace = FALSE)

**Arguments**
x  Vector of elements to sample from.
n  Size of the sample.
replace  Sample with replacement.

**Details**
Rcpp implementation of the sample function.

**Value**
vector

**See Also**
sample
**reconstruction**

**Description**

Randomized data for species b using pattern reconstruction.

**Usage**

```r
reconstruction
```

**Format**

```r
rd_pat object.
```

---

**reconstruct_pattern_cluster**

**Description**

Pattern reconstruction for clustered patterns

**Usage**

```r
reconstruct_pattern_cluster(pattern, n_random = 1, e_threshold = 0.01, 
max_runs = 1000, no_change = Inf, annealing = 0.01, 
comp_fast = 1000, weights = c(0.5, 0.5), r_length = 250, 
return_input = TRUE, simplify = FALSE, verbose = TRUE, 
plot = FALSE)
```

**Arguments**

- `pattern` ppp.
- `n_random` Number of randomizations.
- `e_threshold` Minimum energy to stop reconstruction.
- `max_runs` Maximum number of iterations of `e_threshold` is not reached.
- `no_change` Reconstruction will stop if energy does not decrease for this number of iterations.
- `annealing` Probability to keep relocated point even if energy did not decrease.
- `comp_fast` If pattern contains more points than threshold, summary functions are estimated in a computational fast way.
- `weights` Weights used to calculate energy. The first number refers to Gest(r), the second number to pcf(r).
The functions randomizes the observed pattern by using pattern reconstruction as described in Tscheschel & Stoyan (2006) and Wiegand & Moloney (2014). The algorithm starts with a random but clustered pattern, shifts a point to a new location and keeps the change only, if the deviation between the observed and the reconstructed pattern decreases. The pair correlation function and the nearest neighbour distance function are used to describe the patterns.

For large patterns (n > \text{comp_fast}) the pair correlation function can be estimated from Ripley’s K-function without edge correction. This decreases the computational time. For more information see \text{estimate_pcf_fast}.

The reconstruction can be stopped automatically if for n steps the energy does not decrease. The number of steps can be controlled by \text{no_change} and is set to \text{no_change} = \text{Inf} as default to never stop automatically.

The weights must be 0 < \text{sum(weights)} <= 1. To weight both summary functions identical, use weights = c(0.5, 0.5).

\text{spatstat} sets \text{r_length} to 513 by default. However, a lower value decreases the computational time while increasing the "bumpiness" of the summary function.

### References


### See Also

- \text{calculate_energy}
- \text{plot_randomized_pattern}
- \text{reconstruct_pattern_homo}
- \text{reconstruct_pattern_hetero}
- \text{reconstruct_pattern_marks}
Examples

## Not run:
```r
pattern_recon <- reconstruct_pattern_cluster(species_b, n_random = 19, max_runs = 1000)
```
## End(Not run)

---

**reconstruct_pattern_hetero**

**Usage**

`reconstruct_pattern_hetero(pattern, n_random = 1, e_threshold = 0.01, max_runs = 1000, no_change = Inf, annealing = 0.01, comp_fast = 1000, weights = c(0.5, 0.5), r_length = 250, return_input = TRUE, simplify = FALSE, verbose = TRUE, plot = FALSE)`

**Arguments**

- **pattern**: `ppp`
- **n_random**: Number of randomizations.
- **e_threshold**: Minimum energy to stop reconstruction.
- **max_runs**: Maximum number of iterations of `e_threshold` is not reached.
- **no_change**: Reconstruction will stop if energy does not decrease for this number of iterations.
- **annealing**: Probability to keep relocated point even if energy did not decrease.
- **comp_fast**: If pattern contains more points than threshold, summary functions are estimated in a computational fast way.
- **weights**: Weights used to calculate energy. The first number refers to `Gest(r)`, the second number to `pcf(r)`.
- **r_length**: Number of intervals from `r = 0` to `r = rmax` the summary functions are evaluated.
- **return_input**: The original input data is returned as last list entry.
- **simplify**: If `n_random = 1` and `return_input = FALSE` only pattern will be returned.
- **verbose**: Print progress report.
- **plot**: Plot `pcf` function during optimization.
Details

The functions randomizes the observed pattern by using pattern reconstruction as described in Tscheschel & Stoyan (2006) and Wiegand & Moloney (2014). The algorithm starts with a random but heterogenous pattern, shifts a point to a new location and keeps the change only, if the deviation between the observed and the reconstructed pattern decreases. The pair correlation function and the nearest neighbour distance function are used to describe the patterns.

For large patterns (n > comp_fast) the pair correlation function can be estimated from Ripley’s K-function without edge correction. This decreases the computational time. For more information see estimate_pcf_fast.

The reconstruction can be stopped automatically if for n steps the energy does not decrease. The number of steps can be controlled by no_change and is set to no_change = Inf as default to never stop automatically.

The weights must be 0 < sum(weights) <= 1. To weight both summary functions identical, use weights = c(0.5, 0.5).

spatstat sets r_length to 513 by default. However, a lower value decreases the computational time while increasing the "bumpiness" of the summary function.

Value

list

References


See Also

calculate_energy
plot_randomized_pattern reconstruct_pattern_homo
reconstruct_pattern_cluster
reconstruct_pattern_marks

Examples

## Not run:
input_pattern <- spatstat::rpoispp(lambda = function(x, y) {100 * exp(-3 * x)}, nsim = 1)

pattern_recon <- reconstruct_pattern_hetero(input_pattern, n_random = 19, max_runs = 1000)

## End(Not run)
**reconstruct_pattern_homo**

**Description**

Pattern reconstruction

**Usage**

```r
tests::test_that("reconstruct_pattern_homo",
  reconstruct_pattern_homo(pattern, n_random = 1, e_threshold = 0.01,
    max_runs = 1000, no_change = Inf, annealing = 0.01,
    comp_fast = 1000, weights = c(0.5, 0.5), r_length = 250,
    return_input = TRUE, simplify = FALSE, verbose = TRUE,
    plot = FALSE)
```

**Arguments**

- `pattern` `ppp.
- `n_random` Number of randomizations.
- `e_threshold` Minimum energy to stop reconstruction.
- `max_runs` Maximum number of iterations of `e_threshold` is not reached.
- `no_change` Reconstruction will stop if energy does not decrease for this number of iterations.
- `annealing` Probability to keep relocated point even if energy did not decrease.
- `comp_fast` If pattern contains more points than threshold, summary functions are estimated in a computational fast way.
- `weights` Weights used to calculate energy. The first number refers to `Gest(r)`, the second number to `pcf(r)`.
- `r_length` Number of intervals from \( r = 0 \) to \( r = r_{max} \) the summary functions are evaluated.
- `return_input` The original input data is returned as last list entry.
- `simplify` If `n_random = 1` and `return_input = FALSE` only pattern will be returned.
- `verbose` Print progress report.
- `plot` Plot `pcf` function during optimization.

**Details**

The functions randomizes the observed pattern by using pattern reconstruction as described in Tscheschel & Stoyan (2006) and Wiegand & Moloney (2014). The algorithm starts with a random pattern, shifts a point to a new location and keeps the change only, if the deviation between the observed and the reconstructed pattern decreases. The pair correlation function and the nearest neighbour distance function are used to describe the patterns.
For large patterns \((n > \text{comp\_fast})\) the pair correlation function can be estimated from Ripley’s K-function without edge correction. This decreases the computational time. For more information see \texttt{estimate\_pcf\_fast}.

The reconstruction can be stopped automatically if for \(n\) steps the energy does not decrease. The number of steps can be controlled by \texttt{no\_change} and is set to \texttt{no\_change = Inf} as default to never stop automatically.

The weights must be \(0 < \text{sum(weights)} \leq 1\). To weight both summary functions identical, use \(\text{weights = c(0.5,0.5)}\).

\texttt{spatstat} sets \texttt{r\_length} to 513 by default. However, a lower value decreases the computational time while increasing the “bumpiness” of the summary function.

\section*{Value}
list

\section*{References}


\section*{See Also}
\texttt{calculate\_energy}
\texttt{plot\_randomized\_pattern}
\texttt{reconstruct\_pattern\_hetero}
\texttt{reconstruct\_pattern\_cluster}
\texttt{reconstruct\_pattern\_marks}

\section*{Examples}
\begin{verbatim}
## Not run:
pattern_recon <- reconstruct_pattern_homo(species_a, n_random = 19, max_runs = 1000)
## End(Not run)
\end{verbatim}

\section*{Description}
Pattern reconstruction of marks
reconstruct_pattern_marks

Usage

reconstruct_pattern_marks(pattern, marked_pattern, n_random = 1,
                         e_threshold = 0.01, max_runs = 10000, no_change = Inf,
                         annealing = 0.01, r_length = 250, return_input = TRUE,
                         simplify = FALSE, verbose = TRUE, plot = FALSE)

Arguments

- **pattern**: ppp.
- **marked_pattern**: ppp (marked; see details).
- **n_random**: Number of randomizations.
- **e_threshold**: Minimum energy to stop reconstruction.
- **max_runs**: Maximum number of iterations of e_threshold is not reached.
- **no_change**: Reconstruction will stop if energy does not decrease for this number of iterations.
- **annealing**: Probability to keep relocated point even if energy did not decrease.
- **r_length**: Number of intervals from r = 0 to r = rmax the summary functions are evaluated.
- **return_input**: The original input data is returned as last list entry.
- **simplify**: If n_random = 1 and return_input = FALSE only pattern will be returned.
- **verbose**: Print progress report.
- **plot**: Plot kmmr function during optimization.

Details

The function randomizes the numeric marks of a point pattern using pattern reconstruction as described in Tscheschel & Stoyan (2006) and Wiegand & Moloney (2014). Therefore, an unmarked as well as a marked pattern must be provided. The unmarked pattern must have the spatial characteristics and the same observation window and number of points as the marked one (see `reconstruct_pattern` or `fit_point_process`). Marks must be numeric because the mark-correlation function is used as summary function. Two randomly chosen marks are switch each iterations and changes only kept if the deviation between the observed and the reconstructed pattern decreases.

spatstat sets r_length to 513 by default. However, a lower value decreases the computational time while increasing the "bumpiness" of the summary function.

Value

list

References


See Also

- `fit_point_process`
- `reconstruct_pattern_homo`
- `reconstruct_pattern_hetero`
- `reconstruct_pattern_cluster`

Examples

```r
## Not run:
pattern_recon <- reconstruct_pattern_homo(species_a, n_random = 1, max_runs = 1000, simplify = TRUE, return_input = FALSE)
marks_sub <- spatstat::subset.ppp(species_a, select = dbh)
marks_recon <- reconstruct_pattern_marks(pattern_recon, marks_sub, n_random = 19, max_runs = 1000)
## End(Not run)
```

Description

Results habitat association

Usage

```r
results_habitat_association(pattern, raster, significance_level = 0.05, verbose = TRUE)
```

Arguments

- `pattern` Point pattern or list with reconstructed patterns.
- `raster` RasterLayer or list of RasterLayers.
- `significance_level` Significance level
- `verbose` Print output

Details

The functions shows significant habitat associations by comparing the number of points within a habitat between the observed data and randomized data as described in Plotkin et al. (2000) and Harms et al. (2001). Significant positive or associations are present if the observed count in a habitat is above or below a certain threshold of the randomized count, respectively.

Value

- `data.frame`
References


See Also

- randomize_raster
- translate_raster
- reconstruct_pattern_homo
- reconstruct_pattern_hetero
- reconstruct_pattern_cluster

Examples

```r
landscape_classified <- classify_habitats(landscape, classes = 5)
species_a_random <- fit_point_process(species_a, n_random = 199)
results_habitat_association(pattern = species_a_random, raster = landscape_classified)
```

Description

Analyse species-habitat associations in R. Therefore, information about the location of the species is needed and about the environmental conditions. To test for significance habitat associations, one of the two components is randomized. Methods are mainly based on Plotkin et al. (2000) <doi:10.1006/jtbi.2000.2158> and Harms et al. (2001) <doi:10.1111/j.1365-2745.2001.00615.x>.

Author(s)

Maintainer: Maximillian H.K. Hesselbarth <maximilian.hesselbarth@uni-goettingen.de> (0000-0003-1125-9918)

Authors:

- Marco Sciaini <sciaini.marco@gmail.com> (0000-0002-3042-5435)

See Also

Useful links:

- [https://r-spatialecology.github.io/shar](https://r-spatialecology.github.io/shar)
<table>
<thead>
<tr>
<th>Species</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>species_a</td>
<td>A species with negative associations to habitat 4 of ‘landscape’.</td>
</tr>
<tr>
<td>species_b</td>
<td>A species with positive associations to habitat 5 of ‘landscape’.</td>
</tr>
<tr>
<td>torus_trans</td>
<td>Torus translation of the classified landscape data.</td>
</tr>
</tbody>
</table>

**Usage**

species_a

species_b

torus_trans

**Format**

A spatstat ppp object.

rd_ras object.
**translate_raster**

### Description

Torus translation

### Usage

```r
translate_raster(raster, steps_x = NULL, steps_y = NULL,
                 return_input = TRUE, simplify = FALSE, verbose = TRUE)
```

### Arguments

- `raster`: RasterLayer.
- `steps_x`, `steps_y`: Number of steps (cells) the raster is translated into the corresponding direction. If both are null, all possible combinations are used.
- `return_input`: The original input data is returned as last list entry.
- `simplify`: If n_random = 1 and return_input = FALSE only raster will be returned.
- `verbose`: Print progress report.

### Details

Torus translation test as described in Harms et al. (20001). The raster is shifted in all four cardinal directions by steps equal to the raste resolution. If a cell exits the extent on one side, it enters the extent on the opposite side.

### Value

list

### References


### See Also

`randomize_raster`
### Not run:

```r
landscape_classified <- classify_habitats(landscape, classes = 5)

landscape_random <- translate_raster(landscape_classified)
landscape_random_sub <- translate_raster(landscape_classified, steps_x = 1:10, steps_y = 1:5)
```

## End(Not run)
Index

*Topic **datasets**
  - gamma_test, 8
  - landscape, 9
  - random_walk, 16
  - reconstruction, 17
  - species_a, 26
  - species_b, 26
  - torus_trans, 26

adjacent, 15

calculate_energy, 2, 9, 18, 20, 22
classify_habitats, 4
classIntervals, 4
create_neighbourhood, 5

estimate_pcf_fast, 3, 6, 10, 18, 20, 22
extract_points, 7

fit_point_process, 7, 13, 24

gamma_test, 8

Kest, 6

landscape, 9

pcf.fv, 6
plot_energy, 3, 9
plot_randomized_pattern, 3, 9, 10, 18, 20, 22
plot_randomized_raster, 11
print.rd_mar, 12
print.rd_pat, 13
print.rd_ras, 14

random_walk, 16
randomize_raster, 5, 14, 14, 25, 27
rcpp_sample, 16
reconstruct_pattern_cluster, 3, 9, 13, 17, 20, 22, 24, 25
reconstruct_pattern_hetero, 3, 9, 13, 18, 19, 22, 24, 25
reconstruct_pattern_homo, 3, 9, 13, 18, 20, 21, 24, 25
reconstruct_pattern_marks, 12, 18, 20, 22, 22
reconstruction, 17
results_habitat_association, 24

sample, 16
shar, 25
shar-package (shar), 25
species_a, 26
species_b, 26
torus_trans, 26
translate_raster, 15, 25, 27