Package ‘SESraster’

January 20, 2025

Title Raster Randomization for Null Hypothesis Testing

Version 0.7.1

Description Randomization of presence/absence species distribution raster data with or without including spatial structure for calculating standardized effect sizes and testing null hypothesis. The randomization algorithms are based on classical algorithms for matrices (Gotelli 2000, <doi:10.2307/177478>) implemented for raster data.

License GPL (>= 3)

URL `https://CRAN.R-project.org/package=SESraster`,
`https://github.com/HemingNM/SESraster`,
`https://hemingnm.github.io/SESraster/`

BugReports `https://github.com/HemingNM/SESraster/issues`

Depends R (>= 2.10)

Imports graphics, methods, rlang, stats, terra, utils

Suggests kableExtra, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

Author Neander Marcel Heming [aut, cre, cph]
(<`https://orcid.org/0000-0003-2461-5045`>),
Flávio M. M. Mota [aut] (<`https://orcid.org/0000-0002-0308-7151`>),
Gabriela Alves-Ferreira [aut] (<`https://orcid.org/0000-0001-5661-3381`>)

Maintainer Neander Marcel Heming <`neanderh@yahoo.com.br`>

Repository CRAN

Date/Publication 2024-10-22 12:10:02 UTC

Contents

`algorithm_metrics` *Performance of randomization algorithms*

Description

Compares the richness and occurrence incidence across species between actual and randomized species distributions

Usage

```
algorithm_metrics(
  x,
  spat_alg = NULL,
  spat_alg_args = NULL,
  aleats = 10,
  filename = "",
  force_wr_aleat_file = FALSE,
  ...
)
```

Arguments

`x`	SpatRaster. A SpatRaster containing presence-absence data (0 or 1) for a set of species.
`spat_alg`	A function with the algorithm implementing the desired randomization method. It must work with SpatRaster objects. See examples. Example of functions that work are: `bootspat_naive`, `bootspat_str`, `bootspat_ff`.
`spat_alg_args`	List of arguments passed to the randomization method chosen in 'spat_alg'. See `bootspat_naive`, `bootspat_str`, `bootspat_ff`
`aleats`	positive integer. A positive integer indicating how many times the calculation should be repeated.
`filename`	character. Output filename

`force_wr_aleat_file`
: logical. Force writing bootstrapped rasters, even if files fit in memory. Mostly used for internal test units.

`...`
: additional arguments passed to 'terra::app()' function.

Value

a list with two components:

- spp_metrics: a matrix with metrics comparing actual and randomized frequency of species occurrence. Metrics are average, sd, min, and max frequency across randomizations, sp_reldiff (average difference relative to species frequency), global_reldiff (average difference relative to the number of available cells), upper and lower confidence intervals for sp_reldiff and global_reldiff.
- spat_rich_diff: a SpatRaster with summary statistics about differences between actual and bootstrapped site (cell) richness

Author(s)

Neander M. Heming

See Also

`bootspat_str`, `bootspat_naive`, `bootspat_ff`, `SESraster`, `plot_alg_metrics`

Examples

```
library(SESraster)
library(terra)
r <- load_ext_data()
algorithm_metrics(r, spat_alg = "bootspat_naive", spat_alg_args=list(random="species"), aleats = 3)
algorithm_metrics(r, spat_alg = "bootspat_naive", spat_alg_args=list(random="site"), aleats = 3)
# algorithm_metrics(r, spat_alg = "bootspat_naive", spat_alg_args=list(random="both"))
```

`bootspat_ff` *Spatially structured fixed-fixed sample*

Description

Randomizes a raster stack with fixed richness and species frequency of incidence. Randomizations are based on frequencies (given or calculated from x) and, optionally, a probability raster stack. The probability raster stack controls the probability that a given species is sampled in each cell raster. Frequency controls the number of cells being sampled for each species.

Usage

```
bootspat_ff(
  x,
  rprob = NULL,
  rich = NULL,
  fr = NULL,
  glob_fr = NULL,
  cores = 1,
  filename = "",
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	SpatRaster. A presence-absence SpatRaster.
`rprob`	SpatRaster. Stack of probability values. Structures the spatial pattern of each randomized species.
`rich`	SpatRaster. Richness pattern structuring the sample size of each cell randomization. Calculated if not provided.
`fr`	The observed frequency of incidence (i.e. number of occupied pixels) of each species is across the study area.
`glob_fr`	The size (i.e. number of pixels) of the study area.
`cores`	positive integer. If `cores > 1`, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional parameters for terra::app

Details

The algorithm is based on the algorithm of Connor & Simberloff (1979). It takes each species at a time and placed on Nj (species frequency of incidence) randomly chosen sites (cells). The original algorithm randomly chooses the sequence of species and fills sites (originally islands) until they reach the observed species richness. However, as sites (cells) are filled with species, some species do not have enough available sites to be placed, and their sampled frequency is smaller than observed. Additionally, some sites cannot be completely filled because duplicated species are not allowed in the same site. Their solution was to increase the number of sites to place the species. Here, we opted to order the sequence of species from the largest Nj to the smallest. Also, the probability of occupying a site is given by cell expected richness and on each round (i.e. species placement), the expected richness of newly occupied sites is reduced. This ensures that there will be available sites for all species and the randomized frequency of incidence equals the observed frequency of incidence (Nj).

Value

SpatRaster object

Author(s)

Neander Marcel Heming

References

Connor, E. F., & Simberloff, D. (1979). The Assembly of Species Communities: Chance or Competition? Ecology, 60(6), 1132–1140.

See Also

bootspat_str, bootspat_naive, SESraster, algorithm_metrics

Examples

```
# load random species distributions
library(SESraster)
library(terra)
r <- load_ext_data()
plot(r)

# applying the function
rand.str <- bootspat_str(r)
plot(rand.str)

# With null probability raster
rprobnull <- terra::app(r,
                       function(x){
                       ifelse(is.na(x), NA, 1)
                       })
rand.str2 <- bootspat_str(r, rprob = rprobnull)


library(SESraster)
library(terra)
# creating random species distributions
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
set.seed(510)
r10 <- rast(lapply(1:8,
                function(i, r, mn, mx){
                  app(r, function(x, t){
                    sapply(x, function(x, t){
                       x<max(t) & x>min(t)
                    }, t=t)
                  }, t=sample(seq(mn, mx), 2))
                }, r=r, mn=minmax(r)[1]+10, mx=minmax(r)[2]-10))

names(r10) <- paste("sp", 1:nlyr(r10))
```

```
plot(r10)

rprobnull <- terra::app(r10,
                       function(x){
                       ifelse(is.na(x), NA, 1)
                       })

# bootstrapping once
randr10 <- bootspat_ff(r10, rprobnull)
plot(randr10)
plot(c(sum(r10), sum(randr10)), main=c("observed", "randomized"))
plot(sum(r10)-sum(randr10))
cbind(observed=sapply(r10, function(x)freq(x)[2,3]),
      randomized=sapply(randr10, function(x)freq(x)[2,3]))
```

bootspat_naive *Randomize a set of rasters according to the observed frequency.*

Description

Randomize a set of rasters according to the observed frequency using the methods: sites (by cells), species (by layer) or both (layers and cells). The randomization not assign values to cells with nodata.

Usage

```
bootspat_naive(
  x,
  random = c("site", "species", "both"),
  filename = "",
  memory = NULL,
  cores = 1,
  ...
)
```

Arguments

`x`	SpatRaster. A presence-absence SpatRaster.
`random`	character. Character indicating the type of randomization to be used. The available types are by "site", "specie" or "both". The first method (site) keeps species richness constant within each site (cell)pixel by randomizing the position (presence/absence) of the species within each cell of the stack.
`filename`	character. Output filename
`memory`	logical.
`cores`	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`...`	additional arguments to be passed passed down from a calling function.

Details

The first method (site) is performed within each site (cell) by randomizing the position (presence/absence) of the species within each cell of the stack. This method keeps species richness constant at each cell but the size of the species distribution might change. The second method (species) is performed at each layer (species) of the stack by randomizing the position of species presences in space. This method changes the species richness at each cell but the size of the species distribution is held constant (except if randomization is performed by frequency). The third method (both) combines randomization by site and species at the same time. This method will shuffle all presences across cells and layers, changing site richness and species distribution sizes and location at the same time.

Value

SpatRaster object

Author(s)

Neander Marcel Heming and Gabriela Alves-Ferreira

See Also

`bootspat_str`, `bootspat_ff`, `SESraster`, `algorithm_metrics`

Examples

```
library(terra)
# load random species distributions
r <- load_ext_data()
plot(r)

# randomize pres/abs data by site
rn <- bootspat_naive(r, "site")
plot(rn)


library(SESraster)
library(terra)
# creating random species distributions
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
set.seed(510)
r10 <- rast(lapply(1:18,
                function(i, r, mn, mx){
                  app(r, function(x, t){
                    sapply(x, function(x, t){
                       x<max(t) & x>min(t)
                    }, t=t)
                  }, t=sample(seq(mn, mx), 2))
                }, r=r, mn=minmax(r)[1]+10, mx=minmax(r)[2]-10))

names(r10) <- paste("sp", 1:nlyr(r10))
```

```
plot(r10)

# bootstrapping once
randr10 <- bootspat_naive(r10, "site")
plot(randr10)

plot(c(sum(r10), sum(randr10)), main=c("observed", "randomized"))
cbind(observed=sapply(r10, function(x)freq(x)[2,3]),
      randomized=sapply(randr10, function(x)freq(x)[2,3]))
```

bootspat_str *Spatially structured sample*

Description

Randomizes a raster stack with fixed richness. Randomizations are based on frequencies (given or calculated from x) and, optionally, a probability raster stack. Both, frequencies and probability raster stack, control the probability that a given species is sampled in each cell raster. Frequency controls the probability of each species being sampled compared to all others. Probability raster stack controls the probability that each species is sampled in a given raster cell.

Usage

```
bootspat_str(
  x,
  rprob = NULL,
  rich = NULL,
  fr_prob = NULL,
  cores = 1,
  filename = "",
  memory = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	SpatRaster. A presence-absence SpatRaster.
`rprob`	SpatRaster. Stack of probability values. Structures the spatial pattern of each randomized species.
`rich`	SpatRaster. Richness pattern structuring the sample size of each cell randomization. Calculated if not provided.
`fr_prob`	Either frequency of pixels or probability that a species is observed across the whole layer.

`cores`	positive integer. If `cores` > 1, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`filename`	character. Output filename
`memory`	logical. Checks if there is enough available RAM memory. Calculated if NULL
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`...`	additional parameters for terra::app

Value

SpatRaster object

Author(s)

Neander Marcel Heming

See Also

`bootspat_naive`, `bootspat_ff`, `SESraster`, `algorithm_metrics`

Examples

```
# load random species distributions
library(SESraster)
library(terra)
r <- load_ext_data()
plot(r)

# applying the function
rand.str <- bootspat_str(r)
plot(rand.str)

# With null probability raster
rprobnull <- terra::app(r,
                       function(x){
                       ifelse(is.na(x), NA, 1)
                       })
rand.str2 <- bootspat_str(r, rprob = rprobnull)


library(SESraster)
library(terra)
# creating random species distributions
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
set.seed(510)
r10 <- rast(lapply(1:18,
                function(i, r, mn, mx){
                  app(r, function(x, t){
                    sapply(x, function(x, t){
                       x<max(t) & x>min(t)
```

```
                        }, t=t)
                      }, t=sample(seq(mn, mx), 2))
                    }, r=r, mn=minmax(r)[1]+10, mx=minmax(r)[2]-10))

names(r10) <- paste("sp", 1:nlyr(r10))
plot(r10)

rprobnull <- terra::app(r10,
                       function(x){
                       ifelse(is.na(x), NA, 1)
                       })

# bootstrapping once
randr10 <- bootspat_str(r10, rprobnull)
plot(randr10)
plot(c(sum(r10), sum(randr10)), main=c("observed", "randomized"))
cbind(observed=sapply(r10, function(x)freq(x)[2,3]),
      randomized=sapply(randr10, function(x)freq(x)[2,3]))
```

fit.memory — *Function to evaluate if the rasters generated in the function fit on RAM memory*

Description

Function to evaluate if the rasters generated in the function fit on RAM memory

Usage

```
fit.memory(x, n = 1)
```

Arguments

x	SpatRaster
n	positive integer. The number of copies of x that are needed

Value

logical

Author(s)

Neander Marcel Heming and Gabriela Alves-Ferreira

fr2prob — *Adjust probability of sampling based on frequency of occurrences.*

Description

This function is used to adjust the probability of a species to be sampled across the raster, so that the sampled frequency of occurrence of the species is closer to the observed

Usage

```
fr2prob(x, rprob = NULL)
```

Arguments

`x`	SpatRaster. A presence-absence raster (stack).
`rprob`	SpatRaster. A raster (stack) of probabilities.

Value

numeric vector

Examples

```
library(SESraster)
library(terra)
# load random species distributions
r <- load_ext_data()

# applying the function
fr2prob(r)


f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
set.seed(510)
r10 <- rast(lapply(1:18,
                function(i, r, mn, mx){
                  app(r, function(x, t){
                    sapply(x, function(x, t){
                       x<max(t) & x>min(t)
                    }, t=t)
                  }, t=sample(seq(mn, mx), 2))
                }, r=r, mn=minmax(r)[1]+10, mx=minmax(r)[2]-10))

names(r10) <- paste("sp", 1:nlyr(r10))
fr2prob(r10)
# raw frequencies
unlist(terra::global(r10, function(x)sum(x, na.rm=TRUE)))
```

`load_ext_data` *Load SESraster external datasets*

Description

This function loads external datasets available at extdata package folder

Usage

```
load_ext_data(x = "spp_sites")
```

Arguments

x	dataset to be loaded

Details

These are the available datasets:

- spp_sites: a SpatRaster with randomly generated presence-absence data for five species.

Value

SpatRaster object

Examples

```
# load random species distributions
library(SESraster)
library(terra)

r <- load_ext_data()

plot(r)
```

`plot_alg_metrics` *Plot performance of randomization algorithms*

Description

Plots objects returned by `algorithm_metrics`

Usage

```
plot_alg_metrics(x, what = "spp", ...)
```

Arguments

`x`	list. Object returned by `algorithm_metrics`
`what`	What should be plotted, "species" or "site" metrics?
`...`	Additional parameters passed to `plot`

Author(s)

Neander M. Heming

See Also

`algorithm_metrics`

Examples

```
library(SESraster)
library(terra)
r <- load_ext_data()
am1 <- algorithm_metrics(r, spat_alg = "bootspat_naive", spat_alg_args=list(random="species"))
am2 <- algorithm_metrics(r, spat_alg = "bootspat_naive", spat_alg_args=list(random="site"))
plot_alg_metrics(am1)
plot_alg_metrics(am2)
plot_alg_metrics(am1, "site")
```

SESraster *Standardized effect sizes for SpatRaster objects*

Description

Calculates the standardized effect sizes using a custom function and a null model algorithm.

Usage

```
SESraster(
  x,
  FUN = NULL,
  FUN_args = list(),
  spat_alg = NULL,
  spat_alg_args = list(),
  Fa_sample = NULL,
  Fa_alg = NULL,
  Fa_alg_args = list(),
  aleats = 10,
  cores = 1,
  filename = "",
  overwrite = FALSE,
  force_wr_aleat_file = FALSE,
  ...
)
```

Arguments

`x`	SpatRaster. A SpatRaster containing presence-absence data (0 or 1) for a set of species.
`FUN`	The function to be applied. It must work with SpatRaster objects. See examples.
`FUN_args`	Named list of arguments passed to the FUN
`spat_alg`	A function with the algorithm implementing the desired randomization method. It must work with SpatRaster objects. See examples. Example of functions that work are: `bootspat_naive`, `bootspat_str`, `bootspat_ff`.
`spat_alg_args`	List of arguments passed to the randomization method chosen in 'spat_alg'. See `bootspat_naive`, `bootspat_str`, `bootspat_ff`
`Fa_sample`	Named list of length 1 with a FUN argument (e.g. a vector) to be randomized
`Fa_alg`	function to randomize any non spatial argument to be passed to 'FUN'.
`Fa_alg_args`	Named list of arguments passed to the function in 'Fa_alg'
`aleats`	positive integer. A positive integer indicating how many times the calculation should be repeated.
`cores`	positive integer. If `cores` > `1`, a 'parallel' package cluster with that many cores is created and used. You can also supply a cluster object. Ignored for functions that are implemented by terra in C++ (see under fun)
`filename`	character. Output filename
`overwrite`	logical. If `TRUE`, `filename` is overwritten
`force_wr_aleat_file`	logical. Force writing bootstrapped rasters, even if files fit in memory. Mostly used for internal test units.
`...`	additional arguments passed to 'terra::app()' function.

Details

Perform n=aleats spatial randomizations based on the randomization method defined in 'spat_alg' argument and calculates the metric defined in 'FUN' argument. The function (FUN) to calculate the desired metric must work with any of app, focal, focal3D family of functions.

Value

SpatRaster. The function returns the observed metric, the mean of the simulations calculated over n=aleats times, the standard deviation of the simulations, the standardized effect size (SES) for the metric defined in FUN, and p values for the upper and lower tails.

Author(s)

Neander M. Heming and Gabriela Alves-Ferreira

References

Gotelli 2000

See Also

bootspat_str, bootspat_naive, bootspat_ff, algorithm_metrics

Examples

```
library(SESraster)
library(terra)
r <- load_ext_data()
appmean <- function(x, ...){
                      terra::app(x, "mean", ...)
                    }
ses <- SESraster(r, FUN=appmean, spat_alg = "bootspat_naive", spat_alg_args=list(random="species"),
                 aleats = 4)
plot(ses)
ses <- SESraster(r, FUN=appmean, spat_alg = "bootspat_naive", spat_alg_args=list(random="site"),
                 aleats = 4)
plot(ses)

## example of how to use 'FUN_args'
r[7][1] <- NA
plot(r)
set.seed(10)
sesNA <- SESraster(r, FUN=appmean, FUN_args = list(na.rm = FALSE),
                 spat_alg = "bootspat_naive", spat_alg_args=list(random = "species"),
                 aleats = 4)
plot(sesNA)

set.seed(10)
ses <- SESraster(r, FUN=appmean, FUN_args = list(na.rm = TRUE),
                spat_alg = "bootspat_naive", spat_alg_args=list(random = "species"),
                 aleats = 4)
plot(ses)

## example with 'Fa_alg'
appsv <- function(x, lyrv, na.rm = FALSE, ...){
                      sumw <- function(x, lyrv, na.rm, ...){
                            ifelse(all(is.na(x)), NA,
                                    sum(x*lyrv, na.rm=na.rm, ...))
                      }
               stats::setNames(terra::app(x, sumw, lyrv = lyrv, na.rm=na.rm, ...), "sumw")
                    }
set.seed(10)
ses <- SESraster(r, FUN=appsv,
                 FUN_args = list(lyrv = seq_len(nlyr(r)), na.rm = TRUE),
                    Fa_sample = "lyrv",
                    Fa_alg = "sample", Fa_alg_args = list(replace=FALSE),
                    aleats = 4)
plot(ses)
```

```
set.seed(10)
ses <- SESraster(r, FUN=appsv,
                 FUN_args = list(lyrv = seq_len(nlyr(r)), na.rm = TRUE),
                    Fa_sample = "lyrv",
                    Fa_alg = "sample", Fa_alg_args = list(replace=TRUE),
                    aleats = 4)
plot(ses)
```

Index