Package 'virtualspecies'

September 25, 2014

Type Package

Title Generation of Virtual Species Distributions

Version 1.0

Date 2014-09-25

Author Boris Leroy with help from C. N. Meynard, C. Bellard & F. Courchamp

Maintainer Boris Leroy <leroy.boris@gmail.com>

Description Provides a framework for generating virtual species distributions,a procedure increasingly used in ecology to improve species distribution models.
This package integrates the existing methodological approaches with the
objective of generating virtual species distributions with increased ecological realism.

License GPL (>= 2.0)

Depends raster

Imports ade4, dismo, rworldmap

URL http://borisleroy.com/virtualspecies

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-25 23:05:19

R topics documented:

`virtualspecies-package`

Generation of virtual species

Description

This package allows generating virtual species distributions, for example for testing species distribution modelling protocols. For a complete tutorial, see http://borisleroy.com/virtualspecies

Details

The process of generating a virtual species distribution is divided into four major steps.

1. Generate a virtual species distributions from environmental variables. This can be done by
   - defining the response functions to each environmental variables with `generateSpFromFun`
   - computing a PCA among environmental variables, and simulating the response of the species along the two first axes of the PCA with `generateSpFromPCA`

   This step can be entirely randomised with `generateRandomSp`
2. Convert the virtual species distribution into presence-absence, with `convertToPA`
3. Facultatively, introduce a distribution bias with `limitDistribution`
4. Sample occurrence points (presence only or presence-absence) inside the virtual species distribution, either randomly or with biases, with `sampleOccurrences`

There are other useful functions in the package:

- `formatFunctions`: this is a helper function to format and illustrate the response functions as a correct input for `generateSpFromFun`
- `plotResponse`: to visualise the species-environment relationship of the virtual species
- `removeCollinearity`: this function can be used to remove collinearity among variables of a stack by selecting a subset of non-intercorrelated variables
- `synchroniseNA`: this function can be used to synchronise NA values among layers of a stack

This packages makes use of different other packages:

- This package makes extensive use of the package `raster` to obtain spatialised environmental variables, and produce spatialised virtual species distributions.
- `dismo` is used to sample occurrence data from the generated virtual species.

- ade4 is used to generate species with a PCA approach.
- rworldmap is used to obtain free world shapefiles, in order to create dispersal limitations and sampling biases.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Maintainer: Boris Leroy <leroy.boris@gmail.com>

`convertToPA` *Convert a virtual species distribution (or a suitability raster) into presence-absence*

Description

This functions converts the probabilities of presence from the output of `generateSpFromFun`, `generateSpFromPCA`, `generateRandomSp` or a suitability raster into a presence-absence raster. The conversion can be threshold-based, or based on a probability of conversion (see details).

Usage

```
convertToPA(x, PA.method = "probability", beta = "random", alpha = -0.05,
  species.prevalence = NULL, plot = TRUE)
```

Arguments

`x`	a suitability raster, or the output from functions `generateSpFromFun`, `generateSpFromPCA` or `generateRandomSp`
`PA.method`	`"threshold"` or `"probability"`. If `"threshold"`, then occurrence probabilities are simply converted into presence-absence according to the threshold `beta`. If `"probability"`, then probabilities are converted according to a logistic function of threshold `beta` and slope `alpha`.
`beta`	`"random"`, a numeric value in the range of your probabilities or `NULL`. This is the threshold of conversion into presence-absence (= the inflexion point if `PA.method = "probability"`). If `"random"`, a numeric value will be randomly generated within the range of `x`.
`alpha`	`NULL` or a negative numeric value. Only useful if `PA.method = "probability"`. The value of `alpha` will shape the logistic function transforming occurrences into presence-absences. See `logisticFun` and examples therein for the choice of `alpha`
`species.prevalence`	`NULL` or a numeric value between 0 and 1. The species prevalence is the proportion of sites actually occupied by the species.
`plot`	`TRUE` or `FALSE`. If `TRUE`, maps of probabilities of occurrence and presence-absence will be plotted.

Details

The conversion of probabilities of occurrence into presence - absence is usually performed by selecting a threshold above which presence always occurs, and never below. However, this approach may be too much unrealistic because species may sometime be present in areas with a low probability of occurrence, or be absent from areas with a high probability of occurrence. In addition, when using a threshold you erase the previously generated response shapes: it all becomes threshold. Thus, this threshold approach should be avoided.

Hence, a more realistic conversion consists in converting probabilities into presence - absence with a probability function (see references). Such a probability conversion can be performed here with a logit function (see `logisticFun`).

To perform the probability conversion you have to define two of the three following parameters:

- `beta`: the 'threshold' of the logistic function (i.e. the inflexion point)
- `alpha`: the slope of the logistic function
- `species.prevalence`: the proportion of sites in which the species occur

If you provide `beta` and `alpha`, the `species.prevalence` is calculated immediately calculated after conversion into presence-absence.

On the other hand, if you provide `species.prevalence` and either `beta` or `alpha`, the function will try to determine `alpha` (if you provided `beta`) or `beta` (if you provided `alpha`).

The relationship between species prevalence, alpha and beta is dependent on the available range of environmental conditions (see Meynard and Kaplan, 2011 and especially the Supporting Information). As a consequence, the desired species prevalence may not be available for the defined `alpha` or `beta`. In these conditions, the function will retain the `alpha` or `beta` which provides the closest prevalence to your `species.prevalence`, but you may also provide another value of `alpha` or `beta` to obtain a closer prevalence.

In all cases, the `species.prevalence` indicated in the output is the prevalence measured on the output presence-absence map.

Value

a `list` containing 5 elements:

- `approach`: the approach used to generate the species, *i.e.*, `"response"`
- `details`: the details and parameters used to generate the species
- `suitab.raster`: the virtual species distribution, as a Raster object containing the environmental suitability)
- `PA.conversion`: the parameters used to convert the suitability into presence-absence
- `pa.raster`: the presence-absence map, as a Raster object containing 0 (absence) / 1 (presence) / NA

The structure of the virtualspecies object can be seen using str()

Note

The approximation of `alpha` or `beta` to the chosen `species.prevalence` may take time if you work on very large rasters.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

References

Meynard C.N. & Kaplan D.M. 2013. Using virtual species to study species distributions and model performance. *Journal of Biogeography* **40**:1-8

Meynard C.N. & Kaplan D.M. 2011. The effect of a gradual response to the environment on species distribution model performance. *Ecography* **35**:499-509

Examples

```
# Create an example stack with two environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100))
names(env) <- c("variable1", "variable2")

# Creation of the parameter list
parameters <- formatFunctions(variable1 = c(fun = 'dnorm', mean = 0.00012,
                                            sd = 0.0001),
                              variable2 = c(fun = 'linearFun', a = 1, b = 0))
sp1 <- generateSpFromFun(env, parameters, plot = FALSE)

# Conversion into presence-absence with a threshold-based approach
convertToPA(sp1, PA.method = "threshold", beta = 0.2,  plot = TRUE)
convertToPA(sp1, PA.method = "threshold", beta = "random", plot = TRUE)

# Conversion into presence-absence with a probability-based approach
convertToPA(sp1, PA.method = "probability", beta = 0.4,
              alpha = -0.05, plot = TRUE)
convertToPA(sp1, PA.method = "probability", beta = "random",
              alpha = -0.1, plot = TRUE)

# Conversion into presence-absence by choosing the prevalence
# Threshold method
convertToPA(sp1, PA.method = "threshold",
              species.prevalence = 0.3, plot = TRUE)
# Probability method, with alpha provided
convertToPA(sp1, PA.method = "probability", alpha = -0.1,
              species.prevalence = 0.2, plot = TRUE)
# Probability method, with beta provided
convertToPA(sp1, PA.method = "probability", beta = 0.5,
              species.prevalence = 0.2, alpha = NULL,
              plot = TRUE)

# Plot the output Presence-Absence raster only
sp1 <- convertToPA(sp1, plot = FALSE)
plot(sp1$pa.raster)
```

formatFunctions *Format and visualise functions used to generate virtual species with* generateSpFromFun

Description

This function is a helper function to simplify the formatting of functions for generateSpFromFun.

Usage

```
formatFunctions(x = NULL, rescale = TRUE, ...)
```

Arguments

x	NULL or a RasterStack. If you want to visualise the functions, provide your RasterStack here.
rescale	TRUE or FALSE. If TRUE, individual response plots are rescaled between 0 and 1.
...	the parameters to be formatted. See details.

Details

This function formats the parameters argument of generateSpFromFun. For each environmental variable, provide a vector containing the function name, and its arguments.

For example, assume we want to generate a species responding to two environmental variables bio1 and bio2.

- The response to bio1 is a normal response (dnorm), of mean 1 and standard deviation 0.5.
- The response to bio2 is a linear response (linearFun), of slope (a) 2 and intercept (b) 5.

The correct writing is:

```
formatFunctions( bio1 = c(fun = "dnorm", mean = 1, sd = 0.5),bio2 = c(fun = "linearFun", a = 2, b = 5))
```

Warning

Do not use 'x' as a name for your environmental variables.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
my.parameters <- formatFunctions(variable1 = c(fun = 'dnorm',
                                               mean = 0.00012, sd = 0.0001),
                                 variable2 = c(fun = 'linearFun', a = 1, b = 0))


my.parameters <- formatFunctions(bio1 = c(fun = "logisticFun",
                                          alpha = -12.7, beta = 68),
                                 bio2 = c(fun = "linearFun",
                                          a = -0.03, b = 191.2),
                                 bio3 = c(fun = "dnorm",
                                          mean = 86.4, sd = 19.1),
                                 bio4 = c(fun = "logisticFun",
                                          alpha = 2198.5, beta = 11381.4))
## Not run:
# An example using worldclim data
bio1.4 <- getData('worldclim', var='bio', res=10)[[1:4]]
my.parameters <- formatFunctions(x = bio1.4,
                                 bio1 = c(fun = "logisticFun",
                                          alpha = -12.7, beta = 68),
                                 bio2 = c(fun = "linearFun",
                                          a = -0.03, b = 191.2),
                                 bio3 = c(fun = "dnorm",
                                          mean = 86.4, sd = 19.1),
                                 bio4 = c(fun = "logisticFun",
                                          alpha = 2198.5, beta = 11381.4))

## End(Not run)
```

`generateRandomSp` *Generate a random virtual species distribution from environmental variables*

Description

This function generates randomly a virtual species distribution.

Usage

```
generateRandomSp(raster.stack, approach = "automatic", rescale = TRUE,
  convert.to.PA = TRUE, relations = c("gaussian", "linear", "logistic",
  "quadratic"), rescale.each.response = TRUE, realistic.sp = TRUE,
  species.type = "multiplicative", niche.breadth = "any",
  sample.points = FALSE, nb.points = 10000, PA.method = "probability",
  alpha = -0.1, beta = "random", species.prevalence = NULL, plot = TRUE)
```

Arguments

raster.stack	a RasterStack object, in which each layer represent an environmental variable.
approach	"automatic", "random", "response" or "pca". This parameters defines how species will be generated. "automatic": If less than 6 variables in raster.stack, a response approach will be used, otherwise a pca approach will be used. "random": the approach will be randomly picked. Otherwise choose "response" or "pca". See details.
rescale	TRUE or FALSE. If TRUE, the final probability of presence is rescaled between 0 and 1.
convert.to.PA	TRUE or FALSE. If TRUE, the virtual species distribution will also be converted into Presence-Absence.
relations	[response approach] a vector containing the possible types of response function. The implemented type of relations are "gaussian", "linear", "logistic" and "quadratic".
rescale.each.response	TRUE or FALSE. If TRUE, the individual responses to each environmental variable are rescaled between 0 and 1
realistic.sp	[response approach] TRUE or FALSE. If TRUE, the function will try to define responses that can form a viable species. If FALSE, the responses will be randomly generated (may result in environmental conditions that do not co-exist).
species.type	[response approach] "additive", "multiplicative" or "mixed". Defines how the final probability of presence is calculated: if "additive", responses to each variable are summed; if "multiplicative", responses are multiplicated; if "mixed" the final probability will be a random combination of products and sums of individual responses. See generateSpFromFun
niche.breadth	[pca approach] "any", "narrow" or "wide". This parameter defines how tolerant is the species regarding environmental conditions by adjusting the standard deviations of the gaussian functions. See generateSpFromPCA
sample.points	[pca approach] TRUE of FALSE. If you have a large raster file then use this parameter to sample a number of points equal to nb.points.
nb.points	[pca approach] a numeric value. Only useful if sample.points = TRUE. The number of sampled points from the raster, to perform the PCA. A too small value may not be representative of the environmental conditions in your raster.
PA.method	"threshold" or "probability". If "threshold", then occurrence probabilities are simply converted into presence-absence according to the threshold beta. If "probability", then probabilities are converted according to a logistic function of threshold beta and slope alpha.
alpha	NULL or a negative numeric value. Only useful if PA.method = "probability". The value of alpha will shape the logistic function transforming occurrences into presence-absences. See logisticFun and examples therein for the choice of alpha
beta	"random", a numeric value in the range of your probabilities or NULL. This is the threshold of conversion into presence-absence (= the inflexion point if PA.method = "probability"). If "random", a numeric value will be randomly generated within the range of probabilities of occurrence. See convertToPA

species.prevalence
: NULL or a numeric value between 0 and 1. The species prevalence is the proportion of sites actually occupied by the species. See convertToPA

plot
: TRUE or FALSE. If TRUE, the generated virtual species will be plotted.

Details

By default, the function will perform a probabilistic conversion into presence- absence, with a randomly chosen beta threshold. If you want to custmose the conversion parameters, you have to define **two** of the three following parameters:

- beta: the 'threshold' of the logistic function (i.e. the inflexion point)
- alpha: the slope of the logistic function
- species.prevalence: the proportion of sites in which the species occur

If you provide beta and alpha, the species.prevalence is calculated immediately calculated after conversion into presence-absence.

As explained in convertToPA, if you choose choose a precise species.prevalence, it may not be possible to reach this particular value because of the availability of environmental conditions. Several runs may be necessary to reach the desired species.prevalence.

Value

a list with 3 to 5 elements (depending if the conversion to presence-absence was performed):

- approach: the approach used to generate the species, *i.e.*, "response"
- details: the details and parameters used to generate the species
- suitab.raster: the virtual species distribution, as a Raster object containing the environmental suitability)
- PA.conversion: the parameters used to convert the suitability into presence-absence
- pa.raster: the presence-absence map, as a Raster object containing 0 (absence) / 1 (presence) / NA

The structure of the virtualspecies can object be seen using str()

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Create an example stack with six environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)),
```

```
            raster(exp(a)),
            raster(log(a)))
names(env) <- paste("Var", 1:6, sep = "")

# More than 6 variables: by default a PCA approach will be used
generateRandomSp(env)

# Manually choosing a response approach
generateRandomSp(env, approach = "response")

# Randomly choosing the approach
generateRandomSp(env, approach = "random")
```

`generateSpFromFun` *Generate a virtual species distributions with responses to environmental variables*

Description

This function generates a virtual species distribution from a RasterStack of environmental variables and a defined set of responses to each environmental parameter.

Usage

```
generateSpFromFun(raster.stack, parameters, rescale = TRUE,
  species.type = "multiplicative", formula = NULL,
  rescale.each.response = TRUE, plot = FALSE)
```

Arguments

`raster.stack`	a RasterStack object, in which each layer represent an environmental variable.
`parameters`	a list containing the functions of response of the species to environmental variables with their parameters. See details.
`rescale`	`TRUE` or `FALSE`. If `TRUE`, the final probability of presence is rescaled between 0 and 1.
`species.type`	`"additive"`, `"multiplicative"` or `"mixed"`. Defines how the final probability of presence is calculated: if `"additive"`, responses to each variable are summed; if `"multiplicative"`, responses are multiplicated; if `"mixed"` responses are both summed and multiplicated depending on argument `formula`
`formula`	`NULL` to create a random formula to calculate the final probability of presence, or a character string of the form: `"layername1 + layername2 * layername3 * etc."` to manually define it. Only used if `species.type` is set to `"mixed"`
`rescale.each.response`	`TRUE` or `FALSE`. If `TRUE`, the individual responses to each environmental variable are rescaled between 0 and 1 (see details).
`plot`	`TRUE` or `FALSE`. If `TRUE`, the generated virtual species will be plotted.

Details

This functions proceeds into several steps:

1. The response to each environmental variable is calculated with the functions provided in `parameters`. This results in a probability of presence for each variable.
2. If `rescale.each.response` is `TRUE`, each probability of presence is rescaled between 0 and 1.
3. The final probability of presence is calculated according to the chosen `species.type`.
4. If `rescale` is `TRUE`, the final probability of presence is rescaled between 0 and 1.

The RasterStack containing environmental variables must have consistent names, because they will be checked with the `parameters`. For example, they can be named var1, var2, etc. Names can be checked and set with `names(my.stack)`.

The `parameters` have to be carefully created, otherwise the function will not work:

- Either see `formatFunctions` to easily create your list of parameters
- Or create a `list` defined according to the following template:
  `list( var1 = list(fun = 'fun1', args = list(arg1 = ..., arg2 = ..., etc.)), var2 =`
  It is important to keep the same names in the parameters as in the stack of environmental variables. Similarly, argument names must be identical to argument names in the associated function (e.g., if you use `fun = 'dnorm'`, then args should look like `list(mean = 0, sd = 1)`).
  See the example section below for more examples.

Any response function that can be applied to the environmental variables can be chosen here. Several functions are proposed in this package: `linearFun`, `logisticFun` and `quadraticFun`. Another classical example is the normal distribution: `dnorm`. Ther users can also create and use their own functions.

If `rescale.each.response = TRUE`, then the probability response to each variable will be normalised between 0 and 1 according to the following formula: P.rescaled = (P - min(P)) / (max(P) - min (P)) This rescaling has a strong impact on response functions, so users may prefer to use `rescale.each.response = FALSE` and apply their own rescaling within their response functions.

Value

a `list` with 3 elements:

- `approach`: the approach used to generate the species, *i.e.*, `"response"`
- `details`: the details and parameters used to generate the species
- `suitab.raster`: the virtual species distribution, as a Raster object containing the environmental suitability)

#' The structure of the virtualspecies object can be seen using str()

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

See Also

generateSpFromPCA to generate a virtual species with a PCA approach

Examples

```
# Create an example stack with two environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100))
names(env) <- c("variable1", "variable2")
plot(env) # Illustration of the variables

# Easy creation of the parameter list:
# see in real time the shape of the response functions
parameters <- formatFunctions(variable1 = c(fun = 'dnorm', mean = 1e-04,
                                            sd = 1e-04),
                              variable2 = c(fun = 'linearFun', a = 1, b = 0))

# If you provide env, then you can see the shape of response functions:
parameters <- formatFunctions(x = env,
                              variable1 = c(fun = 'dnorm', mean = 1e-04,
                                            sd = 1e-04),
                              variable2 = c(fun = 'linearFun', a = 1, b = 0))

# Generation of the virtual species
sp1 <- generateSpFromFun(env, parameters)
sp1
par(mfrow = c(1, 1))
plot(sp1)


# Manual creation of the parameter list
# Note that the variable names are the same as above
parameters <- list(variable1 = list(fun = 'dnorm',
                                    args = list(mean = 0.00012,
                                                sd = 0.0001)),
                   variable2 = list(fun = 'linearFun',
                                    args = list(a = 1, b = 0)))
# Generation of the virtual species
sp1 <- generateSpFromFun(env, parameters, plot = TRUE)
sp1
plot(sp1)
```

---

generateSpFromPCA *Generate a virtual species distribution with a PCA of environmental variables*

Description

This functions generates a virtual species distribution by computing a PCA among environmental variables, and simulating the response of the species along the two first axes of the PCA. The response to axes of the PCA is determined with gaussian functions.

Usage

```
generateSpFromPCA(raster.stack, rescale = TRUE, niche.breadth = "any",
  means = NULL, sds = NULL, pca = NULL, remove.collinearity = FALSE,
  multicollinearity.cutoff = 0.7, sample.points = FALSE,
  nb.points = 10000, plot = TRUE)
```

Arguments

raster.stack	a RasterStack object, in which each layer represent an environmental variable.
rescale	TRUE of FALSE. Should the output suitability raster be rescaled between 0 and 1?
niche.breadth	"any", "narrow" or "wide". This parameter defines how tolerant is the species regarding environmental conditions by adjusting the standard deviations of the gaussian functions. See details.
means	a vector containing two numeric values. Will be used to define the means of the gaussian response functions to the axes of the PCA.
sds	a vector containing two numeric values. Will be used to define the standard deviations of the gaussian response functions to the axes of the PCA.
pca	a dudi.pca object. You can provide a pca object that you computed yourself with dudi.pca
remove.collinearity	TRUE of FALSE. Can be set to TRUE to remove groups of collinear variables (at the cutoff multicollinearity.cutoff) in your dataset: only one variable per group will be kept. See removeCollinearity for details.
multicollinearity.cutoff	a numeric value between 0 and 1. Only useful if remove.collinearity = TRUE. The cutoff of collinearity above which variables will be grouped together.
sample.points	TRUE of FALSE. If you have a large raster file then use this parameter to sample a number of points equal to nb.points.
nb.points	a numeric value. Only useful if sample.points = TRUE. The number of sampled points from the raster, to perform the PCA. A too small value may not be representative of the environmental conditions in your raster.
plot	TRUE or FALSE. If TRUE, the generated virtual species will be plotted.

Details

This function proceeds in 3 steps:

1. A PCA of environmental conditions is generated
2. Gaussian responses to the first two axes are computed
3. These responses are multiplied to obtain the final environmental suitability

The shape of gaussian responses can be determined manually, by defining both `means` and `sds`, or randomly. The random generation is constrained by the argument `niche.breadth`, which controls the range of possible standard deviation values. This range of values is based on a fraction of the axis:

- `"any"`: the standard deviations can have values from 1/100 to 1/2 of the axes' span
- `"narrow"`: the standard deviations can have values from 1/100 to 1/10 of the axes' span
- `"wide"`: the standard deviations can have values from 1/10 to 1/2 of the axes' span

Value

a `list` with 3 elements:

- `approach`: the approach used to generate the species, *i.e.*, `"pca"`
- `details`: the details and parameters used to generate the species
- `suitab.raster`: the virtual species distribution, as a Raster object containing the environmental suitability

The structure of the virtualspecies object can be seen using str()

Note

To perform the PCA, the function has to transform the raster into a matrix. This may not be feasible if the raster is too large for the computer's memory. In this case, you should perform the PCA on a sample of your raster with set `sample.points = TRUE` and choose the number of points to sample with `nb.points`.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

See Also

`generateSpFromFun` to generate a virtual species with the responses to each environmental variables. #'

Examples

```
# Create an example stack with four environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)))
names(env) <- c("var1", "var2", "var3", "var4")
plot(env) # Illustration of the variables
```

```
# Generating a species with the PCA

generateSpFromPCA(raster.stack = env)

# The top part of the plot shows the PCA and the response functions along
# the two axes.
# The bottom part shows the probabilities of occurrence of the virtual
# species.




# Defining manually the response to axes

generateSpFromPCA(raster.stack = env,
           means = c(-2, 0),
           sds = c(0.6, 1.5))

# This species can be seen as occupying intermediate altitude ranges of a
# conic mountain.
```

`limitDistribution` *Limit a virtual species distribution to a defined area*

Description

This function is designed to limit species distributions to a subsample of their total distribution range. It will thus generate a species which is not at the equilibrium with its environment (i.e., which did not occupy the full range of suitable environmental conditions).

This function basically takes any type of raster and will limit values above 0 to areas where the species is allowed to disperse.

Usage

```
limitDistribution(x, geographical.limit = "extent", area = NULL,
  plot = TRUE)
```

Arguments

`x`	a `rasterLayer` object composed of 0, 1 and NA, or the output list from `generateSpFromFun`, `generateSpFromPCA` or `generateRandomSp`
`geographical.limit`	`"country"`, `"region"`, `"continent"`, `"polygon"` or `"extent"`. The method used to limit the distribution range: see details.

area	NULL, a character string, a polygon or an extent object. The area in which the distribution range will be limited: see details. If NULL and geographical.limit = "extent", then you will be asked to draw an extent on the map.
plot	TRUE or FALSE. If TRUE, the resulting limited distribution will be plotted.

Details

How the function works:

The function will remove occurrences of the species outside the chosen area:

- NA are kept unchanged
- 0 are kept unchanged
- values > 0 within the limits of area are kept unchanged
- values > 0 outside the limits of area are set to 0

How to define the area in which the range is limited:

You can choose to limit the distribution range of the species to:

1. a particular country, region or continent (assuming your raster has the WGS84 projection):
   Set the argument geographical.limit to "country", "region" or "continent", and provide the name(s) of the associated countries, regions or continents to area (see examples).
   List of possible area names:
   - Countries: type levels(getMap()@data$SOVEREIGNT) in the console
   - Regions: "Africa", "Antarctica", "Asia", "Australia", "Europe", "North America", "South America"
   - Continents: "Africa", "Antarctica", "Australia", "Eurasia", "North America", "South America"
2. a polygon:
   Set geographical.limit to "polygon", and provide your polygon to area.
3. an extent object:
   Set geographical.limit to "extent", and either provide your extent object to area, or leave it NULL to draw an extent on the map.

Value

a list containing 7 elements:

- approach: the approach used to generate the species, *i.e.*, "response"
- details: the details and parameters used to generate the species
- suitab.raster: the virtual species distribution, as a Raster object containing the environmental suitability)
- PA.conversion: the parameters used to convert the suitability into presence-absence
- pa.raster: the presence-absence map, as a Raster object containing 0 (absence) / 1 (presence) / NA

- geographical.limit: the method used to limit the distribution and the area in which the distribution is restricted
- occupied.area: the area occupied by the virtual species as a Raster of presence-absence

The structure of the virtualspecies object can be seen using str() a list containing:

- geographical.limit: a list with two elements, the method used to limit the distribution and the area in which the distribution is restricted
- occupied.area: the area occupied by the virtual species as a Raster of presence-absence

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Create an example stack with six environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)),
             raster(exp(a)),
             raster(log(a)))
names(env) <- paste("Var", 1:6, sep = "")

# More than 6 variables: by default a PCA approach will be used
sp <- generateRandomSp(env)

# limiting the distribution to a specific extent
limit <- extent(0.5, 0.7, 0.6, 0.8)

limitDistribution(sp, area = limit)
```

---

linearFun *Linear function*

Description

A simple linear function of the form

$$ax + b$$

Usage

```
linearFun(x, a, b)
```

Arguments

x	a numeric value or vector
a	a numeric value or vector
b	a numeric value or vector

Value

a numeric value or vector resulting from the function

Author(s)

Boris Leroy <leroy.boris@gmail.com>

Maintainer: Boris Leroy <leroy.boris@gmail.com>

See Also

logisticFun, quadraticFun

Examples

```
x <- 1:100
y <- linearFun(x, a = 0.5, b = 0)
plot(y ~ x, type = "l")
```

---

logisticFun *Logistic function*

Description

A simple logistic function of the form

$$\frac{1}{1 + e^{\frac{x-\beta}{\alpha}}}$$

Usage

```
logisticFun(x, alpha, beta)
```

Arguments

x	a numeric value or vector
alpha	a numeric value or vector
beta	a numeric value or vector

Details

The value of beta determines the 'threshold' of the logistic curve (i.e. the inflexion point).

The value of alpha determines the slope of the curve (see examples):

- alpha very close to 0 will result in a threshold-like response.
- Values of alpha with the same order of magnitude as the range of x (e.g., the range ofx / 10) will result in a logistic function.
- alpha very far from 0 will result in a linear function.

Value

a numeric value or vector resulting from the function

Author(s)

Boris Leroy <leroy.boris@gmail.com>

Maintainer: Boris Leroy <leroy.boris@gmail.com>

See Also

linearFun, quadraticFun

Examples

```
x <- 1:100
y <- logisticFun(x, alpha = -10, b = 50)
plot(y ~ x, type = "l")

# The effect of alpha:
y1 <- logisticFun(x, alpha = -0.01, b = 50)
y2 <- logisticFun(x, alpha = -10, b = 50)
y3 <- logisticFun(x, alpha = -1000, b = 50)

par(mfrow = c(1, 3))
plot(y1 ~ x, type = "l", main = expression(alpha %->% 0))
plot(y2 ~ x, type = "l", main = expression(alpha %~~% range(x)/10))
plot(y3 ~ x, type = "l", main = expression(alpha %->% infinity))
```

---

plotResponse *Visualise the response of the virtual species to environmental variables*

Description

This function plots the relationships between the virtual species and the environmental variables. It requires either the output from generateSpFromFun, generateSpFromPCA, generateRandomSp, or a manually defined set of environmental variables and response functions.

Usage

```
plotResponse(x, parameters = NULL, approach = NULL, rescale = TRUE, ...)
```

Arguments

x	the output from generateSpFromFun, generateSpFromPCA, generateRandomSp, or a raster layer/stack of environmental variables (see details for the latter).
parameters	in case of manually defined response functions, a list containing the associated parameters. See details.
approach	in case of manually defined response functions, the chosen approach: either "response" for a per-variable response approach, or "pca" for a PCA approach.
rescale	TRUE or FALSE. If TRUE, individual response plots are rescaled between 0 and 1.
...	further arguments to be passed to plot. See plot and par.

Details

If you provide the output from generateSpFromFun, generateSpFromPCA or generateRandomSp then the function will automatically make the appropriate plots.

Otherwise, you can provide a raster layer/stack of environmental variables to x and a list of functions to parameters to perform the plot. In that case, you have to specifiy the approach: "reponse" or "PCA":

- if approach = "response": Provide to parameters a list exactly as defined in generateSpFromFun:
  list(var1 = list(fun = 'fun1', args = list(arg1 = ..., arg2 = ..., etc.)), var2
- if approach = "PCA": Provide to parameters a list containing the following elements:
  - pca: a dudi.pca object computed with dudi.pca
  - means: a vector containing two numeric values. Will be used to define the means of the gaussian response functions to the axes of the PCA.
  - sds a vector containing two numeric values. Will be used to define the standard deviations of the gaussian response functions to the axes of the PCA.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Create an example stack with four environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)))
names(env) <- c("var1", "var2", "var3", "var4")
```

```
# Per-variable response approach:
parameters <- formatFunctions(var1 = c(fun = 'dnorm', mean = 0.00012,
                                       sd = 0.0001),
                              var2 = c(fun = 'linearFun', a = 1, b = 0),
                              var3 = c(fun = 'quadraticFun', a = -20, b = 0.2,
                                       c = 0),
                              var4 = c(fun = 'logisticFun', alpha = -0.001,
                                       beta = 0.002))
sp1 <- generateSpFromFun(env, parameters, plot = TRUE)
plotResponse(sp1)

# PCA approach:
sp2 <- generateSpFromPCA(env, plot = FALSE)
par(mfrow = c(1, 1))
plotResponse(sp2)
```

---

quadraticFun *Quadratic function*

Description

A simple quadratic function of the form

$$ax^2 + bx + c$$

Usage

```
quadraticFun(x, a, b, c)
```

Arguments

x	a numeric value or vector
a	a numeric value or vector
b	a numeric value or vector
c	a numeric value or vector

Value

a numeric value or vector resulting from the function

Author(s)

Boris Leroy <leroy.boris@gmail.com>

Maintainer: Boris Leroy <leroy.boris@gmail.com>

See Also

linearFun, quadraticFun

Examples

```
x <- 1:100
y <- quadraticFun(x, a = 2, b = 2, c = 3)
plot(y ~ x, type = "l")
```

`removeCollinearity` *Remove collinearity among variables of a a raster stack*

Description

This functions analyses the correlation among variables of the provideded stack of environmental variables (using Pearson's R), and can return a vector containing names of variables that are not intercorrelated, or a list containing grouping variables according to their degree of collinearity.

Usage

```
removeCollinearity(raster.stack, multicollinearity.cutoff = 0.7,
  select.variables = FALSE, sample.points = FALSE, nb.points = 10000,
  plot = FALSE)
```

Arguments

`raster.stack`	a RasterStack object, in which each layer represent an environmental variable.
`multicollinearity.cutoff`	a numeric value corresponding to the cutoff of correlation above which to group variables.
`select.variables`	`TRUE` or `FALSE`. If `TRUE`, then the function will choose one variable among each group to return a vector of non correlated variables (see details). If `FALSE`, the function will return a list containing the groups of correlated variables.
`sample.points`	`TRUE` or `FALSE`. If you have a large raster file then use this parameter to sample a number of points equal to `nb.points`.
`nb.points`	a numeric value. Only useful if `sample.points = TRUE`. The number of sampled points from the raster, to perform the PCA. A too small value may not be representative of the environmental conditions in your raster.
`plot`	`TRUE` or `FALSE`. If `TRUE`, the hierarchical ascendant classification used to group variables will be plotted.

Details

This function uses the Pearson's correlation coefficient to analyse correlation among variables. This coefficient is then used to compute a distance matrix, which in turn is used it compute an ascendant hierarchical classification, with the '*complete*' method (see `hclust`). If at least one correlation above the `multicollinearity.cutoff` is detected, then the variables will be grouped according to their degree of correlation.

If `select.variables = TRUE`, then the function will return a vector containing variables that are not intercorrelated. The variables not correlated to any other variables are automatically included in this vector. For each group of intercorrelated variables, one variable will be randomly chosen and included in this vector.

Value

a vector of non correlated variables, or a list where each element is a group of non correlated variables.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Create an example stack with six environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)),
             raster(exp(a)),
             raster(log(a)))
names(env) <- paste("Var", 1:6, sep = "")

# Defaults settings: cutoff at 0.7
removeCollinearity(env, plot = TRUE)

# Changing cutoff to 0.5
removeCollinearity(env, plot = TRUE, multicollinearity.cutoff = 0.5)

# Automatic selection of variables not intercorrelated
removeCollinearity(env, plot = TRUE, select.variables = TRUE)

# Assuming a very large raster file: selecting a subset of points
removeCollinearity(env, plot = TRUE, select.variables = TRUE,
                   sample.points = TRUE, nb.points = 5000)
```

---

sampleOccurrences *Sample occurrences in a virtual species distribution*

Description

This function samples presences within a species distribution, either randomly or with a sampling bias. The sampling bias can be defined manually or with a set of pre-defined biases.

Usage

```
sampleOccurrences(x, n, type = "presence only", sampling.area = NULL,
  detection.probability = 1, correct.by.suitability = FALSE,
  error.probability = 0, bias = "no.bias", bias.strength = 50,
  bias.area = NULL, weights = NULL, sample.prevalence = NULL,
  plot = TRUE)
```

Arguments

x	a rasterLayer object or the output list from generateSpFromFun, generateSpFromPCA, generateRandomSp, convertToPA or limitDistribution The raster must contain values of 0 or 1 (or NA).
n	an integer. The number of occurrence points to sample.
type	"presence only" or "presence-absence". The type of occurrence points to sample.
sampling.area	a character string, a polygon or an extent object. The area in which the sampling will take place. See details.
detection.probability	a numeric value between 0 and 1, corresponding to the probability of detection of the species. See details.
correct.by.suitability	TRUE or FALSE. If TRUE, then the probability of detection will be weighted by the suitability, such that cells with lower suitabilities will further decrease the chance that the species is detected when sampled.
error.probability	TRUE or FALSE. Only useful if type = "presence-absence". Probability to attribute an erroneous presence in cells where the species is absent.
bias	"no.bias", "country", "region", "extent", "polygon" or "manual". The method used to generate a sampling bias: see details.
bias.strength	a positive numeric value. The strength of the bias to be applied in area (as a multiplier). Above 1, area will be oversampled. Below 1, area will be undersampled.
bias.area	NULL, a character string, a polygon or an extent object. The area in which the sampling will be biased: see details. If NULL and bias = "extent", then you will be asked to draw an extent on the map.
weights	NULL or a raster layer. Only used if bias = "manual". The raster of bias weights to be applied to the sampling of occurrences. Higher weights mean a higher probability of sampling.
sample.prevalence	NULL or a numeric value between 0 and 1. Only useful if type = "presence-absence". Defines the sample prevalence, i.e. the proportion of presences sampled. Note that the probabilities of detection and error are applied AFTER this parameter, so the final sample prevalence may not different if you apply probabilities of detection and/or error
plot	TRUE or FALSE. If TRUE, the sampled occurrence points will be plotted.

Details

How the function works:

The function randomly selects n cells in which samples occur. If a `bias` is chosen, then the selection of these cells will be biased according to the type and strength of bias chosen. If the sampling is of `type "presence only"`, then only cells where the species is present will be chosen. If the sampling is of `type "presence-absence"`, then all non-NA cells can be chosen.

The function then samples the species inside the chosen cells. In cells where the species is present the species will always be sampled unless the parameter `detection.probability` is lower than 1. In that case the species will be sampled with the associated probability of detection.

In cells where the species is absent (in case of a `"presence-absence"` sampling), the function will always assign absence unless `error.probability` is greater than 1. In that case, the species can be found present with the associated probability of error. Note that this step happens AFTER the detection step. Hence, in cells where the species is present but not detected, it can still be sampled due to a sampling error.

How to restrict the sampling area:

Use the argument `sampling.area`:

- Provide the name (s) (or a combination of names) of country(ies), region(s) or continent(s). Examples:
  - `sampling.area = "Africa"`
  - `sampling.area = c("Africa", "North America", "France")`
- Provide a polygon (`SpatialPolygons` or `SpatialPolygonsDataFrame` of package `sp`)
- Provide an `extent` object

How the sampling bias works:

The argument `bias.strength` indicates the strength of the bias. For example, a value of 50 will result in 50 times more samples within the `bias.area` than outside. Conversely, a value of 0.5 will result in half less samples within the `bias.area` than outside.

How to choose where the sampling is biased:

You can choose to bias the sampling in:

1. a particular country, region or continent (assuming your raster has the WGS84 projection):

   Set the argument `bias` to `"country"`, `"region"` or `"continent"`, and provide the name(s) of the associated countries, regions or continents to `bias.area` (see examples).

   List of possible `bias.area` names:

   - Countries: type `levels(getMap()@data$SOVEREIGNT)` in the console
   - Regions: "Africa", "Antarctica", "Asia", "Australia", "Europe", "North America", "South America"
   - Continents: "Africa", "Antarctica", "Australia", "Eurasia", "North America", "South America"

2. a polygon:

   Set `bias` to `"polygon"`, and provide your polygon to `area`.

3. an extent object:

   Set `bias` to `"extent"`, and either provide your extent object to `bias.area`, or leave it `NULL` to draw an extent on the map.

Otherwise you can define manually your sampling bias, *e.g.* to create biases along roads. In that case you have to provide to `weights` a raster layer in which each cell contains the probability to be sampled.

Value

a `list` with 3 (unbiased sampling) to 4 (biased sampling) elements:

- `sample.points`: the data.frame containing the coordinates of samples, the real presence-absences (or presence-only) and the sampled presence- absences
- `detection.probability`: the chosen probability of detection of the virtual species
- `error.probability`: the chosen probability to assign presence in cells where the species is absent
- `bias`: if a bias was chosen, then the type of bias and the associated `area` will be included.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Create an example stack with six environmental variables
a <- matrix(rep(dnorm(1:100, 50, sd = 25)),
            nrow = 100, ncol = 100, byrow = TRUE)
env <- stack(raster(a * dnorm(1:100, 50, sd = 25)),
             raster(a * 1:100),
             raster(a * logisticFun(1:100, alpha = 10, beta = 70)),
             raster(t(a)),
             raster(exp(a)),
             raster(log(a)))
names(env) <- paste("Var", 1:6, sep = "")

# More than 6 variables: by default a PCA approach will be used
sp <- generateRandomSp(env, niche.breadth = "wide")

# Sampling of 25 presences
sampleOccurrences(sp, n = 25)

# Sampling of 30 presences and absebces
sampleOccurrences(sp, n = 30, type = "presence-absence")

# Reducing of the probability of detection
sampleOccurrences(sp, n = 30, type = "presence-absence",
                  detection.probability = 0.5)

# Further reducing in relation to environmental suitability
```

```
sampleOccurrences(sp, n = 30, type = "presence-absence",
                  detection.probability = 0.5,
                  correct.by.suitability = TRUE)

# Creating sampling errors (far too much)
sampleOccurrences(sp, n = 30, type = "presence-absence",
                  error.probability = 0.5)

# Introducing a sampling bias (oversampling)
biased.area <- extent(0.5, 0.7, 0.6, 0.8)
sampleOccurrences(sp, n = 50, type = "presence-absence",
                  bias = "extent",
                  bias.area = biased.area)
# Showing the area in which the sampling is biased
plot(biased.area, add = TRUE)

# Introducing a sampling bias (no sampling at all in the chosen area)
biased.area <- extent(0.5, 0.7, 0.6, 0.8)
sampleOccurrences(sp, n = 50, type = "presence-absence",
                  bias = "extent",
                  bias.strength = 0,
                  bias.area = biased.area)
# Showing the area in which the sampling is biased
plot(biased.area, add = TRUE)
```

`synchroniseNA` *Synchronise NA values among layers of a stack*

Description

This function ensures that cells containing NAs are the same among all the layers of a raster stack, i.e.that for any given pixel of the stack, if one layer has a NA, then all layers should be set to NA for that pixel.

Usage

```
synchroniseNA(x)
```

Arguments

`x`	a raster stack object which needs to be synchronised.

Details

This function can do that in two different ways; if your computer has enough RAM a fast way will be used; otherwise a slower but memory-safe way will be used.

Author(s)

Boris Leroy <leroy.boris@gmail.com>

with help from C. N. Meynard, C. Bellard & F. Courchamp

Examples

```
# Creation of a stack with different NAs across layers
m <- matrix(nr = 10, nc = 10, 1:100)
r1 <- raster(m)
r2 <- raster(m)
r1[sample(1:ncell(r1), 20)] <- NA
r2[sample(1:ncell(r2), 20)] <- NA
s <- stack(r1, r2)


# Effect of the synchroniseNA() function
plot(s) # Not yet synchronised
s <- synchroniseNA(s)
plot(s) # Synchronised
```

Index