Package 'dispRity'

July 22, 2025

Title Measuring Disparity

Maintainer Thomas Guillerme <guillert@tcd.ie>

Version 1.9

Date 2024-11-13

Description A modular package for measuring disparity (multidimensional space occupancy). Disparity can be calculated from any matrix defining a multidimensional space. The package provides a set of implemented metrics to measure properties of the space and allows users to provide and test their own metrics. The package also provides functions for looking at disparity in a serial way (e.g. disparity through time) or per groups as well as visualising the results. Finally, this package provides several statistical tests for disparity analysis.

Depends R (>= 3.6.0), ape, stats

Imports ade4, castor, Claddis, ellipse, geometry, GET, graphics, grDevices, MASS, methods, mnormt, parallel, phangorn, phyclust, phylolm, utils, vegan, scales, zoo,

License GPL-3 | file LICENSE

Suggests MCMCglmm, geoscale, testthat, knitr

RoxygenNote 7.2.3

URL https://github.com/TGuillerme/dispRity

NeedsCompilation yes

Author Thomas Guillerme [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-4325-1275>), Mark Puttick [aut, cph], Jack Hatfield [aut, cph]

Repository CRAN

Date/Publication 2024-11-13 18:20:02 UTC

Contents

`dispRity-package` *Measuring Disparity in R*

Description

A modular package for measuring disparity (multidimensional space occupancy). Disparity can be calculated from any matrix defining a multidimensional space. The package provides a set of implemented metrics to measure properties of the space and allows users to provide and test their own metrics (Guillerme (2018) <doi:10.1111/2041-210X.13022>). The package also provides functions for looking at disparity in a serial way (e.g. disparity through time - Guillerme and Cooper (2018) <doi:10.1111/pala.12364>) or per groups as well as visualising the results. Finally, this package provides several statistical tests for disparity analysis.

Author(s)

Thomas Guillerme <guillert@tcd.ie>

`add.tree` *Add, remove or get trees (or subtrees)*

Description

Adding, extracting or removing the tree component from a `dispRity` object.

Usage

```
add.tree(data, tree, replace = FALSE)

get.tree(data, subsets = FALSE, to.root = FALSE)

remove.tree(data)
```

Arguments

data	A dispRity object.
tree	A phylo or mutiPhylo object.
replace	Logical, whether to replace any existing tree (TRUE) or add to it (FALSE; default).
subsets	Either a logical whether to extract the tree for each subset (TRUE) or not (FALSE; default) or specific subset names or numbers.
to.root	Logical, whether to return the subset tree including the root of the tree (TRUE) or only containing the elements in the subset (and their most recent common ancestor; FALSE; default). If data contains time bins (from chrono.subsets with method = "discrete"), and to.root = FALSE it returns the subtrees containing only what's in the bin.

Details

get.tree allows to extract the trees specific to each subsets.

Author(s)

Thomas Guillerme and Jack Hadfield

See Also

custom.subsets, chrono.subsets, boot.matrix, dispRity.

Examples

```
## Loading a dispRity object
data(disparity)
## Loading a tree
data(BeckLee_tree)

## Removing  the tree from the dispRity object
(tree_data <- remove.tree(disparity))

## Extracting the tree
get.tree(tree_data) # is null

## Adding a tree to the disparity object
tree_data <- add.tree(tree_data, tree = BeckLee_tree)

## Extracting the tree
```

```
get.tree(tree_data) # is a "phylo" object

## Adding the same tree again
tree_data <- add.tree(tree_data, tree = BeckLee_tree)
get.tree(tree_data) # is a "multiPhylo" object (2 trees)

## Replacing the two trees by one tree
tree_data <- add.tree(tree_data, tree = BeckLee_tree, replace = TRUE)
get.tree(tree_data) # is a "phylo" object
```

`adonis.dispRity` *adonis dispRity (from* `vegan::adonis2`*)*

Description

Passing `dispRity` objects to the `adonis2` function from the `vegan` package.

Usage

```
adonis.dispRity(
  data,
  formula = matrix ~ group,
  method = "euclidean",
  ...,
  warn = TRUE,
  matrix = 1
)
```

Arguments

`data`	A `dispRity` object with subsets
`formula`	The model formula (default is `matrix ~ group`, see details)
`method`	The distance method to be passed to `adonis2` and eventually to `vegdist` (see details - default `method ="euclidean"`)
`...`	Any optional arguments to be passed to `adonis2`
`warn`	`logical`, whether to print internal warnings (`TRUE`; default - advised) or not (`FALSE`).
`matrix`	`numeric`, which matrix to use (default is `1`).

Details

The first element of the formula (the response) must be called `matrix` and the predictors must be existing in the subsets of the `dispRity` object.

If `data$matrix[[1]]` is not a distance matrix, distance is calculated using the `dist` function. The type of distance can be passed via the standard `method` argument that will be recycled by `adonis2`.

If the dispRity data has custom subsets with a single group, the formula is set to matrix ~ group.

If the dispRity data has custom subsets with multiple group categories (separated by a dot, e.g. c("group1.cat1", "group1.cat2", "group2.catA", "group2.catB") being two groups with two categories each), the default formula is matrix ~ first_group but can be set to any combination (e.g. matrix ~ first_group + second_group).

If the dispRity data has time subsets, the predictor is automatically set to time.

Author(s)

Thomas Guillerme

References

Oksanen J, Simpson G, Blanchet F, Kindt R, Legendre P, Minchin P, O'Hara R, Solymos P, Stevens M, Szoecs E, Wagner H, Barbour M, Bedward M, Bolker B, Borcard D, Carvalho G, Chirico M, De Caceres M, Durand S, Evangelista H, FitzJohn R, Friendly M, Furneaux B, Hannigan G, Hill M, Lahti L, McGlinn D, Ouellette M, Ribeiro Cunha E, Smith T, Stier A, Ter Braak C, Weedon J (2024). vegan: Community Ecology Package_. R package version 2.6-8,

See Also

adonis2, test.dispRity, custom.subsets, chrono.subsets.

Examples

```
## Adonis with one groups

## Generating a random character matrix
character_matrix <- sim.morpho(rtree(20), 50, rates = c(rnorm, 1, 0))
## Calculating the distance matrix
distance_matrix <- as.matrix(dist(character_matrix))
## Creating two groups
random_groups <- list("group1" = 1:10, "group2" = 11:20)

## Generating a dispRity object
random_disparity <- custom.subsets(distance_matrix, random_groups)
## Running a default NPMANOVA
adonis.dispRity(random_disparity)


## Adonis with multiple groups

## Creating a random matrix
random_matrix <- matrix(data = rnorm(90), nrow = 10,
                    dimnames = list(letters[1:10]))
## Creating two groups with two states each
groups <- as.data.frame(matrix(data = c(rep(1,5), rep(2,5), rep(c(1,2), 5)),
         nrow = 10, ncol = 2, dimnames = list(letters[1:10], c("g1", "g2"))))

## Creating the dispRity object
multi_groups <- custom.subsets(random_matrix, groups)
```

```
## Running the NPMANOVA
adonis.dispRity(multi_groups, matrix ~ g1 + g2)

## Adonis with time

## Creating time series
data(BeckLee_mat50)
data(BeckLee_tree)
data(BeckLee_ages)
time_subsets <- chrono.subsets(BeckLee_mat50, BeckLee_tree,
     method = "discrete", inc.nodes = FALSE, time = c(100, 85, 65, 0),
     FADLAD = BeckLee_ages)

## Running the NPMANOVA with time as a predictor
adonis.dispRity(time_subsets, matrix ~ time)

## Running the NPMANOVA with each time bin as a predictor
adonis.dispRity(time_subsets, matrix ~ chrono.subsets)
```

`apply.NA` *Apply inapplicable characters to a matrix.*

Description

Apply inapplicable characters to discrete morphological matrix.

Usage

```
apply.NA(matrix, NAs, tree, invariant = FALSE, verbose = FALSE)
```

Arguments

`matrix`	A discrete morphological matrix.
`NAs`	Either a numeric value of how many characters to make inapplicable or vector of characters inapplicability source (either `"character"` or `"clade"`; see details). The length of this vector must be at maximum half the total number of characters.
`tree`	If any inapplicable source is `"clade"`, a tree from where to select the clades.
`invariant`	Whether to allow invariant sites among the characters with inapplicable data. If `invariant = FALSE` the algorithm will try to remove such characters (if possible).
`verbose`	Whether to be verbose or not.

Details

If the NAs argument is a numeric value n, generates n characters with inapplicable data based on the "clade" source.

The NAs argument intakes a vector of character inapplicability source rendering a number of characters inapplicable using the following sources:

"character" draws inapplicable characters directly from the character matrix, ignoring the phylogeny (i.e. for a random character X, an other random character Y will have inapplicable characters for each character states 0 for character X).

"clade" draws inapplicable characters from the phylogeny: it will randomly apply inapplicable characters states for some characters by randomly selecting clades from the provided tree. The algorithm randomly assigns an inapplicable token for this character for all taxa in this clade or all taxa outside this clade.

For example NAs = c(rep("character", 2), rep("clade", 2)) will generate 4 characters with inapplicable data, two using previous characters and two other using random clades.

Author(s)

Thomas Guillerme

See Also

sim.morpho

Examples

```
set.seed(4)
## A random tree with 15 tips
tree <- rcoal(15)
## setting up the parameters
my_rates = c(rgamma, rate = 10, shape = 5)
my_substitutions = c(runif, 2, 2)

## A Mk matrix (10*50)
matrixMk <- sim.morpho(tree, characters = 100, model = "ER",
     states = c(0.85, 0.15), rates = my_rates, invariant = FALSE)

## Setting the number and source of inapplicable characters
my_inapplicables <- c(rep("character", 5), rep("clade", 5))

## Apply some inapplicable characters to the matrix
matrix <- apply.NA(matrixMk, my_inapplicables, tree, verbose = TRUE)
```

as.covar *as.covar*

Description

Changes a dispRity metric to use the covar element from a dispRity object.

Usage

```
as.covar(fun, ..., VCV = TRUE, loc = FALSE)
```

Arguments

fun	a function to apply to the $covar element of dispRity.
...	any additional arguments to pass to fun.
VCV	logical, whether to use the $VCV component of the elements in dispRity$covar (TRUE; default) or not (FALSE) (see details).
loc	logical, whether to use the $loc component of the elements in dispRity$covar (TRUE) or not (FALSE; default) (see details).

Details

This function effectively transforms the input argument from matrix (or matrix2) to matrix = matrix$VCV and adds a evaluation after the return call to indicate that the function works on a $covar element. Note that if the function does not have an argument called matrix, the first argument is estimated as being the one to be transformed (e.g. if the function has its first argument x, it will transform it to x = x$VCV).

You can toggle between using the $VCV or the $loc argument in the $covar matrix by using either VCV = TRUE, loc = FALSE (to access only fun(matrix = matrix$VCV, ...)), VCV = FALSE, loc = TRUE (to access only matrix = matrix(matrix$loc, nrow = 1), ...) or VCV = TRUE, loc = TRUE (to access fun(matrix = matrix$VCV, loc = matrix$loc, ...); provided fun has an extra loc argument).

For between.groups metrics with matrix and matrix2 arguments, you can provide multiple logicals for VCV and loc to be applied repspectively to matrix and matrix2. For example VCV = TRUE will reinterpret matrix and matrix2 as matrix$VCV and matrix2$VCV but loc = c(TRUE, FALSE) will only reinterpret matrix as matrix$loc (and matrix2 will not be reinterpreted).

Author(s)

Thomas Guillerme

See Also

dispRity MCMCglmm.subsets

Examples

```
## Creating a dispRity
data(charadriiformes)

## Creating a dispRity object from the charadriiformes model
covar_data <- MCMCglmm.subsets(data       = charadriiformes$data,
                               posteriors = charadriiformes$posteriors)

## Get one matrix and one covar matrix
one_matrix <- get.matrix(covar_data, subsets = 1)
one_covar  <- get.covar(covar_data, subsets = 1, n = 1)[[1]][[1]]

## Measure the centroids
centroids(one_matrix)

## Measure the centroids on the covar matrix
as.covar(centroids)(one_covar)
## Is the same as:
centroids(one_covar$VCV)

## Apply the measurement on a dispRity object:
## On the traitspace:
summary(dispRity(covar_data, metric = c(sum, centroids)))
## On the covariance matrices:
summary(dispRity(covar_data, metric = c(sum, as.covar(centroids))))
## The same but with additional options (centre = 100)
summary(dispRity(covar_data,
                 metric = c(sum, as.covar(centroids)),
                 centre = 100))

## Example with the VCV and loc options
## A metric that works with both VCV and loc
## (the sum of the variances minus the distance from the location)
sum.var.dist <- function(matrix, loc = rep(0, ncol(matrix))) {
    ## Get the sum of the diagonal of the matrix
    sum_diag <- sum(diag(matrix))
    ## Get the distance between 0 and the loc
    dist_loc <- dist(matrix(c(rep(0, ncol(matrix)), loc), nrow = 2, byrow = TRUE))[1]
    ## Return the sum of the diagonal minus the distance
    return(sum_diag - dist_loc)
}
## Changing the $loc on one_covar for the demonstration
one_covar$loc <- c(1, 2, 3)
## Metric on the VCV part only
as.covar(sum.var.dist, VCV = TRUE, loc = FALSE)(one_covar)
## Metric on the loc part only
as.covar(sum.var.dist, VCV = FALSE, loc = TRUE)(one_covar)
## Metric on both parts
as.covar(sum.var.dist, VCV = TRUE, loc = TRUE)(one_covar)
```

BeckLee *Beck and Lee 2014 datasets*

Description

Example datasets from Beck and Lee 2014.

Format

three matrices and one phylogenetic tree.

Details

- `BeckLee_tree` A phylogenetic tree with 50 living and fossil taxa
- `BeckLee_mat50` The ordinated matrix based on the 50 taxa cladistic distances
- `BeckLee_mat99` The ordinated matrix based on the 50 taxa + 49 nodes cladistic distances
- `BeckLee_ages` A list of first and last occurrence data for fossil taxa
- `BeckLee_disparity` a `dispRity` object with estimated sum of variances in 120 time bins, boostrapped 100 times from the Beck and Lee data

References

Beck RMD & Lee MSY. 2014. Ancient dates or accelerated rates? Morphological clocks and the antiquity of placental mammals. Proc. R. Soc. B 2014 281 20141278; DOI: 10.1098/rspb.2014.1278

See Also

BeckLee_disparity disparity

BeckLee_disparity *BeckLee_disparity*

Description

An example of a `dispRity` object.

Format

one `dispRity` object.

Details

This matrix is based on the `BeckLee` dataset and split into 120 continuous subsets (`chrono.subsets`). It was bootstrapped 100 times (`boot.matrix`) with four rarefaction levels. Disparity was calculated as the `sum` of the `variances` (`dispRity`).

See Also

BeckLee disparity

Examples

```
## Not run:
## Loading the data
data(BeckLee_mat99)
data(BeckLee_tree)
data(BeckLee_ages)

## Creating the 7 subsets
subsets <- chrono.subsets(BeckLee_mat99, BeckLee_tree,
                          time = seq(from = 0, to = 120, by = 1),
                          method = "continuous", model = "proximity",
                          FADLAD = BeckLee_ages)

## Bootstrapping and rarefying
bootstraps <- boot.matrix(subsets, bootstraps = 100)

## Calculating disparity
BeckLee_disparity <- dispRity(bootstraps, metric = c(sum, variances))

## End(Not run)
```

---

bhatt.coeff *Bhattacharyya Coefficient*

Description

Calculates the probability of overlap between two distributions.

Usage

```
bhatt.coeff(x, y, bw = bw.nrd0, ...)
```

Arguments

x, y	two distributions.
bw	the bandwidth size, either a `numeric` or a `function` (see `bw.nrd0`).
...	optional arguments to be passed to the bw argument.

Author(s)

Thomas Guillerme

References

Bhattacharyya A. **1943**. On a measure of divergence between two statistical populations defined by their probability distributions. Bull. Calcutta Math. Soc., 35, pp. 99-109

See Also

test.dispRity, null.test.

Examples

```
## Two dummy distributions
x <- rnorm(1000, 0, 1)
y <- rnorm(1000, 1, 2)

## What is the probability of overlap of these distributions?
bhatt.coeff(x, y)
```

`boot.matrix` *Bootstraps and rarefies data.*

Description

Bootstraps and rarefies either a matrix or a list of matrices.

Usage

```
boot.matrix(
  data,
  bootstraps = 100,
  boot.type = "full",
  boot.by = "rows",
  rarefaction = FALSE,
  verbose = FALSE,
  prob = NULL
)
```

Arguments

`data`	A `matrix` or a list of matrices (typically output from chrono.subsets or custom.subsets - see details).
`bootstraps`	The number of bootstrap pseudoreplicates (`default = 100`).
`boot.type`	The bootstrap algorithm to use (`default = "full"`; see details).
`boot.by`	Which dimension of the data to bootstrap: either `"rows"` to bootstrap the elements (default), `"columns"` for the dimensions or `"dist"` for bootstrapping both equally (e.g. for distance matrices).

rarefaction	Either a logical value whether to fully rarefy the data, a set of numeric values used to rarefy the data or "min" to rarefy at the minimum level (see details).
verbose	A logical value indicating whether to be verbose or not.
prob	Optional, a matrix or a vector of probabilities for each element to be selected during the bootstrap procedure. The matrix or the vector must have a row names or names attribute that corresponds to the elements in data.

Details

data: The data is considered as the multidimensional space and is not transformed (e.g. if ordinated with negative eigen values, no correction is applied to the matrix).

rarefaction: when the input is numeric, the number of elements is set to the value(s) for each bootstrap. If some subsets have fewer elements than the rarefaction value, the subsets is not rarefied. When the input is "min", the smallest number of elements is used (or 3 if some subsets have less than 3 elements).

boot.type: the different bootstrap algorithms are:

- "full": resamples all the rows of the matrix and replaces them with a new random sample of rows (with replace = TRUE, meaning all the elements can be duplicated in each bootstrap).
- "single": resamples only one row of the matrix and replaces it with a new randomly sampled row (with replace = FALSE, meaning that only one element can be duplicated in each bootstrap).
- "null": resamples all rows of the matrix across subsets. I.e. for each subset of *n* elements, this algorithm resamples *n* elements across *ALL* subsets. If only one subset (or none) is used, this does the same as the "full" algorithm.

prob: This option allows to attribute specific probability to each element to be drawn. A probability of 0 will never sample the element, a probability of 1 will always allow it to be sampled. This can also be useful for weighting elements: an element with a weight of 10 will be sampled ten times more. If the argument is a matrix, it must have rownames attributes corresponding to the element names. If the argument is a vector, it must have names attributes corresponding to the element names.

Multiple trees: If the given data is a chrono.subsets based on multiple trees, the sampling is proportional to the presence of each element in each tree: $\sum(1/n)/T$ (with n being the maximum number of elements among the trees and T being the total numbers of trees). For example, for a slice through two trees resulting in the selection of elements A and B in the first tree and A, B and C in the second tree, the "full" bootstrap algorithm will select three elements (with replacement) between A, B and C with a probability of respectively $p(A) = 1/3$ ($p(A) = (1/3 + 1/3)/2$), $p(B) = 1/3$ and $p(C) = 1/6$ ($p(C) = (0 + 1/3)/2$).

Value

This function outputs a dispRity object containing:

matrix	the multidimensional space (a matrix).
call	A list containing the called arguments.
subsets	A list containing matrices pointing to the elements present in each subsets.

Use summary.dispRity to summarise the dispRity object.

Author(s)

Thomas Guillerme

See Also

cust.subsets, chrono.subsets, dispRity.

Examples

```
## Load the Beck & Lee 2014 matrix
data(BeckLee_mat50)

## Bootstrapping a matrix
## Bootstrapping an ordinated matrix 20 times
boot.matrix(BeckLee_mat50, bootstraps = 20)
## Bootstrapping an ordinated matrix with rarefaction
boot.matrix(BeckLee_mat50, bootstraps = 20, rarefaction = TRUE)
## Bootstrapping an ordinated matrix with only elements 7, 10 and 11 sampled
boot.matrix(BeckLee_mat50, bootstraps = 20, rarefaction = c(7, 10, 11))
## Bootstrapping an the matrix but without sampling Cimolestes and sampling Maelestes 10x more
boot.matrix(BeckLee_mat50, bootstraps = 20, prob = c("Cimolestes" = 0, "Maelestes" = 10))

## Bootstrapping a subsets of matrices
## Generating a dummy subsets of matrices
ordinated_matrix <- matrix(data = rnorm(90), nrow = 10, ncol = 9,
                           dimnames = list(letters[1:10]))
matrix_list <- custom.subsets(ordinated_matrix, list(A = 1:5, B = 6:10))
## Bootstrapping the subsets of matrices 20 times (each)
boot.matrix(matrix_list, bootstraps = 20)
```

---

char.diff *Character differences*

Description

Calculates the character difference from a discrete matrix

Usage

```
char.diff(
  matrix,
  method = "hamming",
  translate = TRUE,
  special.tokens,
  special.behaviours,
  order = FALSE,
  by.col = TRUE,
  correction
)
```

Arguments

matrix	A discrete matrix or a list containing discrete characters. The differences is calculated between the columns (usually characters). Use t(matrix) or by.col = FALSE to calculate the differences between the rows.
method	The method to measure difference: "hamming" (default; Hamming 1950), "manhattan", "comparable", "euclidean", "maximum", "mord" (Lloyd 2016), "none" or "binary".
translate	logical, whether to translate the characters following the *xyz* notation (TRUE - default; see details - Felsenstein 2004) or not (FALSE). Translation works for up to 26 tokens per character.
special.tokens	optional, a named vector of special tokens to be passed to grep (make sure to protect the character with "\\"). By default special.tokens <- c(missing = "\\?", inapplicable = "\\-", polymorphism = "\\&", uncertainty = "\\/"). Note that NA values are not compared and that the symbol "@" is reserved and cannot be used.
special.behaviours	optional, a list of one or more functions for a special behaviour for special.tokens. See details.
order	logical, whether the character should be treated as order (TRUE) or not (FALSE - default). This argument can be a logical vector equivalent to the number of rows or columns in matrix (depending on by.col) to specify ordering for each character.
by.col	logical, whether to measure the distance by columns (TRUE - default) or by rows (FALSE).
correction	optional, an eventual function to apply to the matrix after calculating the distance.

Details

Each method for calculating distance is expressed as a function of $d(x, y)$ where x and y are a pair of columns (if by.col = TRUE) or rows in the matrix and *n* is the number of comparable rows (if by.col = TRUE) or columns between them and *i* is any specific pair of rows (if by.col = TRUE) or columns. The different methods are:

- "hamming" The relative distance between characters. This is equal to the Gower distance for non-numeric comparisons (e.g. character tokens; Gower 1966). $d(x, y) = \sum[i, n](abs(x[i] - y[i])/n$
- "manhattan" The "raw" distance between characters: $d(x, y) = \sum[i, n](abs(x[i] - y[i])$
- "comparable" The number of comparable characters (i.e. the number of tokens that can be compared): $d(x, y) = \sum[i, n]((x[i] - y[i])/(x[i] - y[i]))$
- "euclidean" The euclidean distance between characters: $d(x, y) = \sqrt{(}\sum[i, n]((x[i] - y[i])^2))$
- "maximum" The maximum distance between characters: $d(x, y) = max(abs(x[i] - y[i]))$
- "mord" The maximum observable distance between characters (Lloyd 2016): $d(x, y) = \sum[i, n](abs(x[i] - y[i])/\sum[i, n]((x[i] - y[i])/(x[i] - y[i])$

- "none" Returns the matrix with eventual converted and/or translated tokens.
- "binary" Returns the matrix with the binary characters.

When using translate = TRUE, the characters are translated following the *xyz* notation where the first token is translated to 1, the second to 2, etc. For example, the character 0, 2, 1, 0 is translated to 1, 2, 3, 1. In other words when translate = TRUE, the character tokens are not interpreted as numeric values. When using translate = TRUE, scaled metrics (i.e "hamming" and "gower") are divide by $n - 1$ rather than n due to the first character always being equal to 1.

special.behaviours allows to generate a special rule for the special.tokens. The functions should can take the arguments character, all_states with character being the character that contains the special token and all_states for the character (which is automatically detected by the function). By default, missing data returns and inapplicable returns NA, and polymorphisms and uncertainties return all present states.

- missing = function(x,y) NA
- inapplicable = function(x,y) NA
- polymorphism = function(x,y) strsplit(x, split = "\\&")[[1]]
- uncertainty = function(x,y) strsplit(x, split = "\\/")[[1]]

Functions in the list must be named following the special token of concern (e.g. missing), have only x, y as inputs and a single output a single value (that gets coerced to integer automatically). For example, the special behaviour for the special token "?" can be coded as: special.behaviours = list(missing = function(x, y) return(y) to make all comparisons containing the special token containing "?" return any character state y.

IMPORTANT: Note that for any distance method, NA values are skipped in the distance calculations (e.g. distance(A = {1, NA, 2}, B = {1, 2, 3}) is treated as distance(A = {1, 2}, B = {1, 3})).

IMPORTANT: Note that the number of symbols (tokens) per character is limited by your machine's word-size (32 or 64 bits). If you have more than 64 tokens per character, you might want to use continuous data.

Value

A character difference value or a matrix of class char.diff

Author(s)

Thomas Guillerme

References

Felsenstein, J. **2004**. Inferring phylogenies vol. 2. Sinauer Associates Sunderland. Gower, J.C. **1966**. Some distance properties of latent root and vector methods used in multivariate analysis. Biometrika 53:325-338. Hamming, R.W. **1950**. Error detecting and error correcting codes. The Bell System Technical Journal. DOI: 10.1002/j.1538-7305.1950.tb00463.x. Lloyd, G.T. **2016**. Estimating morphological diversity and tempo with discrete character-taxon matrices: implementation, challenges, progress, and future directions. Biological Journal of the Linnean Society. DOI: 10.1111/bij.12746.

See Also

plot.char.diff, vegdist, dist, calculate_morphological_distances, daisy

Examples

```
## Comparing two binary characters
char.diff(list(c(0, 1, 0, 1), c(0, 1, 1, 1)))

## Pairwise comparisons in a morphological matrix
morpho_matrix <- matrix(sample(c(0,1), 100, replace = TRUE), 10)
char.diff(morpho_matrix)

## Adding special tokens to the matrix
morpho_matrix[sample(1:100, 10)] <- c("?", "0&1", "-")
char.diff(morpho_matrix)

## Modifying special behaviours for tokens with "&" to be treated as NA
char.diff(morpho_matrix,
          special.behaviours = list(polymorphism = function(x,y) return(NA)))

## Adding a special character with a special behaviour (count "%" as "100")
morpho_matrix[sample(1:100, 5)] <- "%"
char.diff(morpho_matrix,
          special.tokens = c("paragraph" = "\\%"),
          special.behaviours = list(paragraph = function(x,y) as.integer(100)))

## Comparing characters with/without translation
char.diff(list(c(0, 1, 0, 1), c(1, 0, 1, 0)), method = "manhattan")
# no character difference
char.diff(list(c(0, 1, 0, 1), c(1, 0, 1, 0)), method = "manhattan",
          translate = FALSE)
# all four character states are different
```

---

charadriiformes *Charadriiformes*

Description

An example of a MCMCglmm model.

Format

one data.frame, one phylo and one MCMCglmm.

Details

This dataset is based on a random subset of 359 Charadriiformes (gulls, plovers and sandpipers) from Cooney et al 2017 and trees from Jetz et al 2012. It contains:

- `data` A `"data.frame"` .
- `tree` A consensus tree of 359 charadriiformes species (`"phylo"`).
- `posteriors` The posteriors from a `"MCMCglmm"` model (see example below).
- `tree_distribution` A random distribution of 10 trees of the 359 charadriiformes species (`"multiPhylo"`).

References

Cooney CR, Bright JA, Capp EJ, Chira AM,Hughes EC, Moody CJ, Nouri LO, Varley ZK, Thomas GH. Mega-evolutionary dynamics of the adaptive radiation of birds. Nature. 2017 Feb;542(7641):344-7.

Jetz W, Thomas GH, Joy JB, Hartmann K, Mooers AO. The global diversity of birds in space and time. Nature. 2012 Nov;491(7424):444-8.

Examples

```
## Not run:
## Reproducing the MCMCglmm model
require(MCMCglmm)
data(charadriiformes)

## Setting up the model parameters:
## 1 - The formula (the first three PC axes)
model_formula <- cbind(PC1, PC2, PC3) ~ trait:clade-1
## 2 - The residual term
model_residuals <- ~us(trait):units
## 3 - The random terms
## (one per clade and one for the whole phylogeny)
model_randoms <- ~ us(at.level(clade,1):trait):animal
                 + us(at.level(clade,2):trait):animal
                 + us(at.level(clade,3):trait):animal
                 + us(trait):animal

## Flat priors for the residuals and random terms
flat_priors <- list(
     ## The residuals priors
     R = list(
         R1 = list(V = diag(3), nu = 0.002)),
     ## The random priors (the phylogenetic terms)
     G = list(
         G1 = list(V = diag(3), nu = 0.002),
         G2 = list(V = diag(3), nu = 0.002),
         G3 = list(V = diag(3), nu = 0.002),
         G4 = list(V = diag(3), nu = 0.002)))

## Run the model for 110000 iterations
## sampled every 100 with a burnin (discard)
## of the first 10000 iterations)
model <- MCMCglmm(formula  = model_formula,
                  rcov     = model_residual,
                  random   = model_randoms,
```

```
                  family   = rep("gaussian", 3),
                  prior    = flat_priors,
                  nitt     = 110000,
                  burnin   = 10000,
                  thin     = 100,
                  pedigree = charadriiformes$tree,
                  data     = charadriiformes$data)

## End(Not run)
```

---

check.morpho *Check a morphological matrix consistency levels.*

Description

Performs a fast check of the phylogenetic signal in a morphological matrix using parsimony.

Usage

```
check.morpho(
  matrix,
  orig.tree,
  parsimony = "fitch",
  first.tree = c(phangorn::dist.hamming, phangorn::NJ),
  distance = phangorn::RF.dist,
  ...,
  contrast.matrix,
  verbose = FALSE
)
```

Arguments

matrix	A discrete morphological matrix.
orig.tree	Optional, the input tree to measure the distance between the parsimony and the original tree.
parsimony	Either the parsimony algorithm to be passed to optim.parsimony or a parsimony function that can take a phyDat object as an input (default = "fitch").
first.tree	A list of functions to generate the first most parsimonious tree (default = c(dist.hamming, NJ); see details).
distance	Optional, if orig.tree is provided, the function to use for measuring distance between the trees (default = link[phangorn]{RF.dist}).
...	Any additional arguments to be passed to the parsimony algorithm.
contrast.matrix	An optional contrast matrix. By default, the function recognises any character state token as different apart from ? that is treated as all characters.
verbose	Whether to be verbose or not (default = FALSE).

Details

- The `first.tree` argument must be a list of functions to be used in a cascade to transform the matrix (as a `phyDat` object) into a tree using the functions iteratively. For example the default `c(dist.hamming, NJ)` will apply the following to the matrix: `NJ(dist.hamming(matrix))`

Value

Returns the parsimony score (using `parsimony`), the consistency and retention indices (using `CI` and `RI`) from the most parsimonious tree obtained from the matrix. Can also return the topological distance from the original tree if provided.

Author(s)

Thomas Guillerme

See Also

`sim.morpho`, `get.contrast.matrix`, `optim.parsimony`

Examples

```
## Generating a random tree
random_tree <- rcoal(10)

## Generating a random matrix
random_matrix <- sim.morpho(random_tree, characters = 50, model = "ER",
     rates = c(rgamma, 1, 1))

## Checking the matrix scores
check.morpho(random_matrix, orig.tree = random_tree)
```

---

chrono.subsets *Separating data in chronological subsets.*

Description

Splits the data into a chronological (time) subsets list.

Usage

```
chrono.subsets(
  data,
  tree = NULL,
  method,
  time,
  model,
  inc.nodes = FALSE,
```

```
  FADLAD = NULL,
  verbose = FALSE,
  t0 = FALSE,
  bind.data = FALSE,
  dist.data = FALSE
)
```

Arguments

data	A matrix or a list of matrices.
tree	NULL (default) or an optional phylo or multiPhylo object matching the data and with a root.time element. This argument can be left missing if method = "discrete" and all elements are present in the optional FADLAD argument.
method	The time subsampling method: either "discrete" (or "d") or "continuous" (or "c").
time	Either a single integer for the number of discrete or continuous samples or a vector containing the age of each sample.
model	One of the following models: "acctran", "deltran", "random", "proximity", "equal.split" or "gradual.split". Is ignored if method = "discrete".
inc.nodes	A logical value indicating whether nodes should be included in the time subsets. Is ignored if method = "continuous".
FADLAD	NULL (default) or an optional data.frame or list of data.frames containing the first and last occurrence data.
verbose	A logical value indicating whether to be verbose or not. Is ignored if method = "discrete".
t0	If time is a number of samples, whether to start the sampling from the tree$root.time (TRUE), or from the first sample containing at least three elements (FALSE - default) or from a fixed time point (if t0 is a single numeric value).
bind.data	If data contains multiple matrices and tree contains the same number of trees, whether to bind the pairs of matrices and the trees (TRUE) or not (FALSE - default).
dist.data	A logical value indicating whether to treat the data as distance data (TRUE) or not (FALSE - default).

Details

The data is considered as the multidimensional space with rows as elements and columns as dimensions and is not transformed (e.g. if ordinated with negative eigen values, no correction is applied to the matrix).

If method = "continuous" and when the sampling is done along an edge of the tree, the data selected for the time subsets can be one of the following:

- Punctuated models:
  - "acctran": always selecting the value from the ancestral node.
  - "deltran": always selecting the value from the descendant node or tip.

  - "random": randomly selecting between the ancestral node or the descendant node/tip.
  - "proximity": selecting the ancestral node or the descendant node/tip with a probability relative to branch length.

- Gradual models:
  - "equal.split": randomly selecting from the ancestral node or the descendant node or tip with a 50% probability each.
  - "gradual.split": selecting the ancestral node or the descendant with a probability relative to branch length.

N.B. "equal.split" and "gradual.split" differ from the punctuated models by outputting a node/tip probability table rather than simply the node and the tip selected. In other words, when bootstrapping using boot.matrix, the two former models will properly integrate the probability to the bootstrap procedure (i.e. different tips/nodes can be drawn) and the two latter models will only use the one node/tip determined by the model before the bootstrapping.

Author(s)

Thomas Guillerme

References

Guillerme T. & Cooper N. **2018**. Time for a rethink: time sub-sampling methods in disparity-through-time analyses. Palaeontology. DOI: 10.1111/pala.12364.

See Also

tree.age, slice.tree, cust.subsets, boot.matrix, dispRity.

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_tree) ; data(BeckLee_mat50)
data(BeckLee_mat99) ; data(BeckLee_ages)

## Time binning (discrete method)
## Generate two discrete time bins from 120 to 40 Ma every 40 Ma
chrono.subsets(data = BeckLee_mat50, tree = BeckLee_tree, method = "discrete",
     time = c(120, 80, 40), inc.nodes = FALSE, FADLAD = BeckLee_ages)
## Generate the same time bins but including nodes
chrono.subsets(data = BeckLee_mat99, tree = BeckLee_tree, method = "discrete",
     time = c(120, 80, 40), inc.nodes = TRUE, FADLAD = BeckLee_ages)

## Time slicing (continuous method)
## Generate five equidistant time slices in the dataset assuming a proximity
## evolutionary model
chrono.subsets(data = BeckLee_mat99, tree = BeckLee_tree,
     method = "continuous", model = "acctran", time = 5,
     FADLAD = BeckLee_ages)
```

`Claddis.ordination` *Imports data from Claddis*

Description

Takes Claddis data and computes both the distance and the ordination matrix

Usage

```
Claddis.ordination(data, distance = "mord", ..., k, add = TRUE, arg.cmdscale)
```

Arguments

`data`	Data from `read_nexus_matrix` or the path to a file to be read by `read.nexus.data` (see details).
`distance`	Distance type to be computed by `calculate_morphological_distances`. Can be either `"gc"`, `"ged"`, `"red"`, `"mord"`. `distance` can also be set to `NULL` to convert a matrix in `read_nexus_matrix` list type (see details).
`...`	Any optional arguments to be passed to `calculate_morphological_distances`.
`k`	The number of dimensions in the ordination. If left empty, the number of dimensions is set to number of rows - 1.
`add`	whether to use the Cailliez correction for negative eigen values (`add = TRUE`; default - see `cmdscale`) or not (`add = FALSE`).
`arg.cmdscale`	Any optional arguments to be passed to `cmdscale` (as a named list such as `list(x.ret = TRUE)`).

Details

If `data` is a file path, the function will use a modified version of `read.nexus.data` (that handles polymorphic and ambiguous characters). The file content will then be converted into a `read_nexus_matrix` type list treating all characters as unordered. If the `distance` is set to `NULL`, `data` will be only converted into a `read_nexus_matrix` type list.

Author(s)

Thomas Guillerme

See Also

`calculate_morphological_distances`, `read_nexus_matrix`, `build_cladistic_matrix`, `cmdscale`, `custom.subsets`, `chrono.subsets`, `boot.matrix`, `dispRity`.

Examples

```
## Not run:
require(Claddis)

## Ordinating the distance matrix of Claddis example data
Claddis.ordination(Claddis::michaux_1989)

## Creating simple discrete morphological matrix (with polymorphisms)
cat(
"#NEXUS
BEGIN DATA;
DIMENSIONS  NTAX=5 NCHAR=5;
FORMAT SYMBOLS= \" 0 1 2\" MISSING=? GAP=- ;
MATRIX
     t1  {01}1010
     t2  02120
     t3  1210(01)
     t4  01111
     t5  00101
;
END;", file = "morpho_matrix.nex")

## Ordinating the matrix (using a distance matrix)
Claddis.ordination("morpho_matrix.nex")

## Only converting the nexus matrix into a Claddis format
Claddis_data <- Claddis.ordination("morpho_matrix.nex", distance = NULL)

file.remove("morpho_matrix.nex")

## End(Not run)
```

`clean.data` *Cleaning phylogenetic data*

Description

Cleans a table/tree to match with a given table/tree

Usage

```
clean.data(data, tree, inc.nodes = FALSE)
```

Arguments

`data`	A `data.frame` or `matrix` with the elements names as row names.
`tree`	A `phylo` or `multiPhylo` object.
`inc.nodes`	Logical, whether to check if the nodes in `tree` are also present in `data` (`TRUE`) or not (`FALSE`; default).

Details

Note if `inc.nodes` is set to `TRUE`, the function outputs an error message if there is no matching.

Value

A `list` containing the cleaned data and tree(s) and information on the eventual dropped tips and rows.

Author(s)

Thomas Guillerme

See Also

`tree.age`.

Examples

```
##Creating a set of different trees
trees_list <- list(rtree(5, tip.label = LETTERS[1:5]), rtree(4,
     tip.label = LETTERS[1:4]), rtree(6, tip.label = LETTERS[1:6]))
class(trees_list) <- "multiPhylo"

##Creating a matrix
dummy_data <- matrix(c(rnorm(5), runif(5)), 5, 2,
    dimnames = list(LETTERS[1:5], c("var1", "var2")))

##Cleaning the trees and the data
cleaned <- clean.data(data = dummy_data, tree = trees_list)
##The taxa that where dropped (tips and rows):
c(cleaned$dropped_tips, cleaned$dropped_rows)
##The cleaned trees:
cleaned$tree
##The cleaned data set:
cleaned$data
```

`covar.plot` *covar.plot*

Description

Visualising components of a `dispRity` object with covar.

Usage

```
covar.plot(
  data,
  n,
  points = TRUE,
  major.axes = FALSE,
  ellipses = FALSE,
  level = 0.95,
  dimensions = c(1, 2),
  centres = colMeans,
  scale,
  transparent.scale,
  add = FALSE,
  apply.to.VCV = FALSE,
  ...
)
```

Arguments

data	an dispRity object with a covar component.
n	optional, a number of random posteriors to use.
points	logical, whether to plot the observed elements (TRUE; default) or not (FALSE).
major.axes	can be either logical for plotting all (or n) major.axes (TRUE) or none (FALSE; default) or a function for displaying one summarised major axis. See details.
ellipses	can be either logical for plotting all (or n) ellipses (TRUE) or none (FALSE; default) or a function for displaying one summarised ellipse. See details.
level	the confidence interval level of the major axes and ellipses (default is 0.95).
dimensions	which dimensions (default is c(1,2)).
centres	optional, a way to determine ellipses or major axes positions. Can be either a function (default is colMeans), a vector or a list of coordinates vectors or "intercept". See details.
scale	optional, the name of a group from data on which to scale the ellipses and major axis to be the same size.
transparent.scale	optional, if multiple major axes and/or ellipses are plotted, a scaling factor for the transparency. If left empty, the transparency is set to 1/n or 0.1 (whichever is higher).
add	logical, whether to add the plot to an existing plot (TRUE) or not (FALSE; default).
apply.to.VCV	logical, if ellipse and/or major.axes is a function, whether to apply it on all the estimated ellipses/major axes (FALSE; default) or on the variance covariance matrices directly (TRUE). In other words, whether to apply the function to the ellipses/major axis or the the VCV first (e.g. the average ellipses or the ellipse of the average VCV).
...	any graphical options to be passed to plot, lines or points. See details.

Details

When specifying optional arguments with `...` in a graph with multiple elements (e.g. `points`, `lines`, etc...) you can specify which specific element to affect using the syntax `<element>.<argument>`. For example if you want everything in the plot to be in blue at the exception of the points to be red, you can use `covar.plot(..., col = "blue", points.col = "red")`.

The arguments `major.axes` and `ellipses` can intake a `function` for summarising the display of multiple variance covariance matrices (if `n` is missing or greater than one). This can be any central tendency function such as `mean`, `median` or `mode.val`.

The argument `centres` allows to determine how to calculate the centre of each ellipses or major axes. The argument can be either:

- A `function` to calculate the centre from a group like the default `colMeans` function that calculates the centroid coordinates of each group;
- A `numeric` value to be replicated as the coordinates for the centre of each group (e.g. `centres = 0` sets all the centres at the coordinates `c(0,0,0,...)`); or a vector of numeric values to be directly used as the coordinates for each group (e.g. `centres = c(1,2,3)` sets all the centres at the coordinates `c(1,2,3)`); or a list of numeric values or numeric vectors to be used as the coordinates for the centres of each group;
- `"intercept"` for using the estimated posterior intercept for each sample.

NOTE that if the input contains more dimensions than the visualised dimensions (by default `dimensions = c(1,2)`) the ellipses and major axes are projected from an n-dimensional space onto a 2D space which might make them look incorrect. *NOTE* also that the ellipses and major axes are measured independently, when summarising both parameters (e.g. by using `ellipses = mean` and `major.axes = mean`), the displayed summarised major axes is not calculated from the summarised ellipse but from the coordinates of all major axes (and therefore might not match the coordinates of the ellipse).

Author(s)

Thomas Guillerme

See Also

`MCMCglmm.subsets` `covar.utilities`

Examples

```
data(charadriiformes)

## Creating a dispRity object from the charadriiformes model
covar <- MCMCglmm.subsets(data       = charadriiformes$data,
                          posteriors = charadriiformes$posteriors,
                          group      = MCMCglmm.levels(
                                           charadriiformes$posteriors)[1:4],
                          rename.groups = c("gulls", "plovers",
                                            "sandpipers", "phylogeny"))

## Default plot
```

```
covar.plot(covar)

## Same plot with more options
covar.plot(covar, n = 50, ellipses = mean, major.axes = TRUE,
           col = c("orange", "blue", "darkgreen", "grey", "grey"),
           legend = TRUE, points = TRUE, points.cex = 0.2,
           main = "Charadriiformes shapespace")
```

`covar.utilities` *Utilities for a dispRity object with covariance matrices*

Description

Different utility functions to extract aspects of a `MCMCglmm` object.

Usage

```
get.covar(data, subsets, sample, n, dimensions)

axis.covar(data, subsets, sample, n, dimensions, level = 0.95, axis = 1)
```

Arguments

`data`	a `dispRity` object with a `covar` element.
`subsets`	optional, a `numeric` or `character` for which subsets to get (if missing, the value for all subsets are given).
`sample`	optional, one or more specific posterior sample IDs (is ignored if n is used) or a function to summarise all axes.
`n`	optional, a random number of covariance matrices to sample (if left empty, all are used).
`dimensions`	optional, which dimensions to use. If missing the dimensions from `data` are used.
`level`	which confidence interval level to use (default is `0.95`).
`axis`	which major axis to calculate (default is `1`, the first one).

Author(s)

Thomas Guillerme

See Also

`MCMCglmm.subsets`

Examples

```
## Load the Charadriiformes dataset
data(charadriiformes)
## Making a dispRity object with covar data
covar_data <- MCMCglmm.subsets(data       = charadriiformes$data,
                               posteriors = charadriiformes$posteriors)

## Get the two first covar matrices for each level
get.covar(covar_data, sample = c(1,2))
## Get 2 random covar matrices in 2D for each level
get.covar(covar_data, n = 2, dimensions = c(1,2))
## Get mean covar matrix for each level
get.covar(covar_data, sample = mean)

## Get the 0.95 major axis for the 42th covar matrix
axis.covar(covar_data, sample = 42)
## Get the 0.5 major axis for 2 random samples
axis.covar(covar_data, n = 1, level = 0.5)
## Get the median 0.95 minor axis of the 2D ellipse
axis.covar(covar_data, sample = mean, dimensions = c(1,2), axis = 2)
```

---

crown.stem *Separates stem and crown species*

Description

Selects the crown

Usage

```
crown.stem(tree, inc.nodes = TRUE, output.names = TRUE)
```

Arguments

tree	a "phylo" object
inc.nodes	whether to include the nodes (TRUE; default) or not (FALSE) in the output.
output.names	whether to output the taxa names (TRUE; default) or two phylogenetic trees (FALSE).

Author(s)

Thomas Guillerme

See Also

custom.subsets, tree.age

Examples

```
## A tree with fossil taxa
data(BeckLee_tree)

## Getting both crown and stem taxa lists
crown.stem(BeckLee_tree)

## Splitting the tree into two subtrees
crown_stem_trees <- crown.stem(BeckLee_tree, output.names = FALSE)
## Graphical parameters
op <- par(mfrow = c(1,3))
## Plotting the trees
plot(BeckLee_tree, main = "Full tree")
plot(crown_stem_trees$crown, main = "Crown group")
plot(crown_stem_trees$stem, main = "Stem group")
```

---

custom.subsets *Separating data into custom subsets.*

Description

Splits the data into a customized subsets list.

Usage

```
custom.subsets(data, group, tree = NULL, dist.data = FALSE)
```

Arguments

data	A matrix or a list of matrices.
group	Either a list of row numbers or names to be used as different groups, a data.frame with the same k elements as in data as rownames, a factor or a logical vector. If group is a phylo object matching data, groups are automatically generated as clades (and the tree is attached to the resulting dispRity object).
tree	NULL (default) or an optional phylo or multiPhylo object to be attached to the data.
dist.data	A logical value indicating whether to treat the data as distance data (TRUE) or not (FALSE - default).

Details

Note that every element from the input data can be assigned to multiple groups!

Author(s)

Thomas Guillerme

See Also

chrono.subsets, boot.matrix, dispRity, crown.stem.

Examples

```
## Generating a dummy ordinated matrix
ordinated_matrix <- matrix(data = rnorm(90), nrow = 10)

## Splitting the ordinated matrix into two groups using row numbers
custom.subsets(ordinated_matrix, list(c(1:4), c(5:10)))

## Splitting the ordinated matrix into three groups using row names
ordinated_matrix <- matrix(data = rnorm(90), nrow = 10,
     dimnames = list(letters[1:10]))
custom.subsets(ordinated_matrix,
     list("A" = c("a", "b", "c", "d"), "B" = c("e", "f", "g", "h", "i", "j"),
          "C" = c("a", "c", "d", "f", "h")))

## Splitting the ordinated matrix into four groups using a dataframe
groups <- as.data.frame(matrix(data = c(rep(1,5), rep(2,5), rep(c(1,2), 5)),
     nrow = 10, ncol = 2, dimnames = list(letters[1:10], c("g1", "g2"))))
custom.subsets(ordinated_matrix, groups)

## Splitting a matrix by clade
data(BeckLee_mat50)
data(BeckLee_tree)
custom.subsets(BeckLee_mat50, group = BeckLee_tree)
```

---

demo_data *Demo datasets*

Description

A set six trait spaces with different groups and different dimensions.

Details

The content of these datasets and the pipeline to build them is described in details in Guillerme et al 2020.

- beck A palaeobiology study of mammals. The data is a 105 dimensions ordination (PCO) of the distances between 106 mammals based on discrete morphological characters.
- wright A palaeobiology study of crinoids. The data is a 41 dimensions ordination (PCO) of the distances between 42 crinoids based on discrete morphological characters.
- marcy A geometric morphometric study of gophers (rodents). The data is a 134 dimensions ordination (PCA) the Procrustes superimposition of landmarks from 454 gopher skulls.

- `hopkins` A geometric morphometric study of trilobites. The data is a 134 dimensions ordination (PCA) the Procrustes superimposition of landmarks from 46 trilobites cephala.
- `jones` An ecological landscape study. The data is a 47 dimensions ordination (PCO) of the Jaccard distances between 48 field sites based on species composition.
- `healy` A life history analysis of the pace of life in animals. The data is a 6 dimensions ordination (PCA) of 6 life history traits from 285 animal species.

Source

doi:10.1002/ece3.6452

References

Guillerme T, Puttick MN, Marcy AE, Weisbecker V. **2020** Shifting spaces: Which disparity or dissimilarity measurement best summarize occupancy in multidimensional spaces?. Ecol Evol. 2020;00:1-16. (doi:10.1002/ece3.6452)

Beck, R. M., & Lee, M. S. (2014). Ancient dates or accelerated rates? Morphological clocks and the antiquity of placental mammals. Proceedings of the Royal Society B: Biological Sciences, 281(1793), 20141278.

Wright, D. F. (2017). Bayesian estimation of fossil phylogenies and the evolution of early to middle Paleozoic crinoids (Echinodermata). Journal of Paleontology, 91(4), 799-814.

Marcy, A. E., Hadly, E. A., Sherratt, E., Garland, K., & Weisbecker, V. (2016). Getting a head in hard soils: convergent skull evolution and divergent allometric patterns explain shape variation in a highly diverse genus of pocket gophers (Thomomys). BMC evolutionary biology, 16(1), 207.

Hopkins, M.J. and Pearson, J.K., 2016. Non-linear ontogenetic shape change in Cryptolithus tesselatus (Trilobita) using three-dimensional geometric morphometrics. Palaeontologia Electronica, 19(3), pp.1-54.

Jones, N. T., Germain, R. M., Grainger, T. N., Hall, A. M., Baldwin, L., & Gilbert, B. (2015). Dispersal mode mediates the effect of patch size and patch connectivity on metacommunity diversity. Journal of Ecology, 103(4), 935-944.

Healy, K., Ezard, T.H., Jones, O.R., Salguero-Gomez, R. and Buckley, Y.M., 2019. Animal life history is shaped by the pace of life and the distribution of age-specific mortality and reproduction. Nature ecology & evolution, p.1.

Examples

```
data(demo_data)

## Loading the Beck and Lee 2014 demo data
demo_data$beck

## Loading the Wright 2017 demo data
demo_data$wright

## Loading the Marcy et al. 2015 demo data
demo_data$marcy

## Loading the Hopkins and Pearson 2016 demo data
```

```
demo_data$hopkins

## Loading the Jones et al. 2015 demo data
demo_data$jones

## Loading the Healy et al. 2019 demo data
demo_data$healy
```

`disparity` *disparity*

Description

An example of a `dispRity` object.

Format

one `dispRity` object.

Details

This matrix is based on the `BeckLee` dataset and split into seven continuous subsets (`chrono.subsets`). It was bootstrapped 100 times (`boot.matrix`) with four rarefaction levels. Disparity was calculated as the `median` of the `centroids` (`dispRity`).

See Also

BeckLee_disparity BeckLee

Examples

```
## Not run:
## Loading the data
data(BeckLee_mat99)
data(BeckLee_tree)
data(BeckLee_ages)

## Creating the 7 subsets
subsets <- chrono.subsets(BeckLee_mat99, BeckLee_tree,
                          time = seq(from = 30, to = 90, by = 10),
                          method = "continuous", model = "ACCTRAN",
                          FADLAD = BeckLee_ages)

## Bootstrapping and rarefying
bootstraps <- boot.matrix(subsets, bootstraps = 100,
                          rarefaction = c(20, 15, 10, 5))

## Calculating disparity
disparity <- dispRity(bootstraps, metric = c(median, centroids))

## End(Not run)
```

dispRity *Calculates disparity from a matrix.*

Description

Calculates disparity from a matrix, a list of matrices or subsets of a matrix, where the disparity metric can be user specified.

Usage

```
dispRity(
  data,
  metric,
  dimensions = NULL,
  ...,
  between.groups = FALSE,
  dist.data = NULL,
  verbose = FALSE,
  tree = NULL
)
```

Arguments

data	A matrix or a dispRity object (see details).
metric	A vector containing one to three functions. At least of must be a dimension-level 1 or 2 function (see details).
dimensions	Optional, a vector of numeric value(s) or the proportion of the dimensions to keep.
...	Optional arguments to be passed to the metric.
between.groups	A logical value indicating whether to run the calculations between groups (TRUE) or not (FALSE - default) or a numeric list of pairs of groups to run (see details).
dist.data	A logical value indicating whether to treat the data as distance data (TRUE) or not (FALSE). By default it is set to NULL and interprets whether to use distance data from data.
verbose	A logical value indicating whether to be verbose or not.
tree	NULL (default) or an optional phylo or multiPhylo object to be attached to the data. If this argument is not null, it will be recycled by metric when possible.

Details

The dispRity object given to the data argument can be: a list of matrices (typically output from the functions chrono.subsets or custom.subsets), a bootstrapped matrix output from boot.matrix, a list of disparity measurements calculated from the dispRity function or a matrix object with rows as elements and columns as dimensions. In any of these cases, the data is considered as the

multidimensional space and is not transformed (e.g. if ordinated with negative eigen values, no correction is applied to the matrix).

`metric` should be input as a vector of functions. The functions are sorted and used by dimension-level from 3 to 1 (see `dispRity.metric` and `make.metric`). Typically dimension-level 3 functions take a `matrix` and output a `matrix`; dimension-level 2 functions take a `matrix` and output a `vector` and dimension-level 1 functions take a `matrix` or a `vector` and output a single value. When more than one function is input, they are treated first by dimension-level (i.e. 3, 2 and finally 1). Note that the functions can only take one metric of each dimension-level and thus can only take a maximum of three arguments!

Some metric functions are built into the `dispRity` package: see `dispRity.metric` For user specified metrics, please use `make.metric` to ensure that the metric will work.

HINT: if using more than three functions you can always create your own function that uses more than one function (e.g. `my_function <- function(matrix) cor(var(matrix))` is perfectly valid and allows one to use two dimension-level 3 functions - the correlation of the variance-covariance matrix in this case).

The `between.groups` argument runs the disparity between groups rather within groups. If `between.groups = TRUE`, the disparity will be calculated using the following inputs:

- if the input is an output from `custom.subsets`, the series are run in a pairwise manner using `metric(matrix, matrix2)`. For example for a `custom.subset` contains 3 subsets m1, m2 and m3, the code loops through: `metric(m1, m2)`, `metric(m2, m3)` and `metric(m1, m3)` (looping through `list(c(1,2), c(2,3), c(3,1))`).
- if the input is an output from `chrono.subsets`, the series are run in a paired series manner using `metric(matrix, matrix2)`. For example for a `chrono.subsets` contains 3 subsets m1, m2, m3 and m4, the code loops through: `metric(m1, m2)` and `metric(m2, m3)` (looping through `list(c(1,2), c(2,3), c(3,4))`).

In both cases it is also possible to specify the input directly by providing the list to loop through. For example using `between.groups = list(c(1,2), c(2,1), c(4,8))` will apply the `metric` to the 1st and 2nd subsets, the 2nd and first and the 4th and 8th (in that specific order).

Value

This function outputs a `dispRity` object containing at least the following:

`matrix`	the multidimensional space (a list of `matrix`).
`call`	A `list` containing the called arguments.
`subsets`	A `list` containing matrices pointing to the elements present in each subsets.
`disparity`	A `list` containing the disparity in each subsets.

Use summary.dispRity to summarise the `dispRity` object.

Author(s)

Thomas Guillerme

See Also

`custom.subsets`, `chrono.subsets`, `boot.matrix`, `dispRity.metric`, `summary.dispRity`, `plot.dispRity`.

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_mat50)

## Calculating the disparity as the sum of variances from a single matrix
sum_of_variances <- dispRity(BeckLee_mat50, metric = c(sum, variances))
summary(sum_of_variances)
## Bootstrapping this value
bootstrapped_data <- boot.matrix(BeckLee_mat50, bootstraps = 100)
dispRity(bootstrapped_data, metric = c(sum, variances))

## Calculating the disparity from a customised subset
## Generating the subsets
customised_subsets <- custom.subsets(BeckLee_mat50,
     list(group1 = 1:(nrow(BeckLee_mat50)/2),
          group2 = (nrow(BeckLee_mat50)/2):nrow(BeckLee_mat50)))
## Bootstrapping the data
bootstrapped_data <- boot.matrix(customised_subsets, bootstraps = 100)
## Calculating the sum of variances
sum_of_variances <- dispRity(bootstrapped_data, metric = c(sum, variances))
summary(sum_of_variances)

## Calculating disparity with different metrics of different dimension-levels
## Disparity is calculated as the distribution of the variances in each
## dimension (output are distributions)
disparity_level2 <- dispRity(BeckLee_mat50, metric = variances)
## Disparity is calculated as the mean of the variances in each dimension
## (output are single values)
disparity_level1 <- dispRity(disparity_level2, metric = mean)
## Both disparities have the same means but dimension-level 1 has no quantiles
summary(disparity_level2)
summary(disparity_level1)
```

`dispRity.covar.projections`

Covar projection analyses wrapper

Description

Wrapper function for a covar projection analyses on dispRity objects

Usage

```
dispRity.covar.projections(
  data,
  type,
  base,
```

```
  sample,
  n,
  major.axis = 1,
  level = 0.95,
  output = c("position", "distance", "degree"),
  inc.base = FALSE,
  ...,
  verbose = FALSE
)
```

Arguments

data	a dispRity object containing a $covar component(e.g. from MCMCglmm.subsets)
type	either "groups" for the projections between groups or "elements" for the projections of elements onto groups.
base	optional, a specific group to project the elements or the groups onto or a list of pairs of groups to compare (see between.groups argument in dispRity). If left empty, the groups are projected onto each other in a pairwise manner and the elements are projected onto their respective groups.
sample	optional, one or more specific posterior sample IDs (is ignored if n is used) or a function to summarise all axes.
n	optional, a random number of covariance matrices to sample (if left empty, all are used).
major.axis	which major axis to use (default is 1; see axis.covar for more details).
level	the confidence interval to estimate the major axis (default is 0.95; see axis.covar for more details)).
output	which values to output from the projection. By default, the three values c("position", "distance", "degree") are used to respectively output the projection, rejection and angle values (see projections for more details). The argument "orthogonality" can also be added to this vector.
inc.base	logical, when using type = "elements" with a supplied base argument, whether to also calculate the projections for the base group (TRUE) or not (FALSE; default).
...	any optional arguments to pass to projections (such as centre or abs). *NOTE that this function uses by default* centre = TRUE *and* abs = TRUE *which are not the defaults for* projections.
verbose	logical, whether to be verbose (TRUE) or not (FALSE, default).

Details

Effectively, the wrapper runs either of the following function (simplified here):

- if type = "groups": dispRity(data, metric = as.covar(projections.between), between.groups = TRUE,) for the projections group in data onto each other.
- if type = "elements": dispRity(data, metric = as.covar(projections), ...) for the projections of each element in data onto their main axis.

If base is specified:

- type = "groups" will run pairs elements each subset and base (instead of the full pairwise analyses).
- type = "elements" will run the projection of each subset onto the major axis from base rather than its own.

Value

A list of class "dispRity" and "projection" which contains dispRity objects corresponding to each projection value from output. The elements of the list can be accessed and analysed individually by selecting them by name (e.g. output$position) or by ID (e.g. output[[1]]). Alternatively, the list can be summarised and plotted using summary.dispRity plot.dispRity.

Author(s)

Thomas Guillerme

References

Guillerme T, Bright JA, Cooney CR, Hughes EC, Varley ZK, Cooper N, Beckerman AP, Thomas GH. 2023. Innovation and elaboration on the avian tree of life. Science Advances. 9(43):eadg1641.

See Also

projections projections.between axis.covar dispRity MCMCglmm.subsets

Examples

```
data(charadriiformes)

## Creating a dispRity object with a covar component
my_covar <-MCMCglmm.subsets(
                data       = charadriiformes$data,
                posteriors = charadriiformes$posteriors,
                tree       = charadriiformes$tree,
                group      = MCMCglmm.levels(
                                charadriiformes$posteriors)[1:4],
                rename.groups = c("gulls", "plovers", "sandpipers", "phylo"))

## Running a projection analyses between groups (on 100 random samples)
between_groups <- dispRity.covar.projections(my_covar, type = "groups", base = "phylo", n = 100)
## Summarising the results
summary(between_groups)

## Measuring the projection of the elements on their own average major axis
elements_proj <- dispRity.covar.projections(my_covar, type = "elements", sample = mean,
                                            output = c("position", "distance"))
## Visualising the results
plot(elements_proj)

## Visualising the correlation
```

```
plot(elements_proj, speicfic.args = list(correlation.plot = c("position", "distance")))
```

`dispRity.fast` *Fast dispRity*

Description

Fast disparity calculations. THIS FUNCTION IS LESS SAFE TO USE THAN `dispRity` (see details).

Usage

```
dispRity.fast(group, space, metric, ...)
```

Arguments

`group`	a logical vector for grouping
`space`	a matrix
`metric`	a metric dispRity style (up to two levels)
`...`	any additional arguments to be passed to `metric`

Details

IN DOUBT, USE `dispRity` *INSTEAD OF THIS FUNCTION.* This function should only be used in very specific cases requiring advanced optimisation or embedded customised functions. This function is simply applying `metric(space[group,])` for each group and returns a list of results. It does not check the validity of the data, metric and groups. It does not handle specific data (e.g. multiple matrices), specific metrics (e.g. no optional arguments), does not produce meaningful error messages and does not intake nor returns a dispRity object.

Author(s)

Thomas Guillerme

Examples

```
## A random space
space <- matrix(rnorm(25), 5, 5)

## A metric
metric <- c(sum, variances)

## A group of four observations
group <- c(TRUE, TRUE, TRUE, TRUE, FALSE)

## The disparity
dispRity.fast(group, space, metric)
```

`dispRity.metric` *Disparity metrics*

Description

Different implemented disparity metrics.

Usage

```
dimension.level3.fun(matrix, ...)
dimension.level2.fun(matrix, ...)
dimension.level1.fun(matrix, ...)
between.groups.fun(matrix, matrix2, ...)
```

Arguments

`matrix`	A matrix.
`...`	Optional arguments to be passed to the function. Usual optional arguments are `method` for specifying the method for calculating distance passed to `vegdist` (e.g. `method = "euclidean"` - default - or `method = "manhattan"`) or `k.root` to scale the result using the kth root. See details below for available optional arguments for each function.
`matrix2`	Optional, a second matrix for metrics between groups.

Details

These are inbuilt functions for calculating disparity. See `make.metric` for details on `dimension.level3.fun`, `dimension.level2.fun`, `dimension.level1.fun` and `between.groups.fun`. The dimensions levels (1, 2 and 3) can be seen as similar to ranks in linear algebra.

The currently implemented dimension-level 1 metrics are:

- `convhull.volume`: calculates the convex hull hypervolume of a matrix (calls `convhulln(x, options = "FA")$vol`).
  - Both `convhull` functions call the `convhulln` function with the `"FA"` option (computes total area and volume).
  - WARNING: both `convhull` functions can be computationally intensive above 10 dimensions!
- `convhull.surface`: calculates the convex hull hypersurface of a matrix (calls `convhulln(x, options = "FA")$area`).
- `diagonal`: calculates the longest distance in the ordinated space.
  - WARNING: This function is the generalisation of Pythagoras' theorem and thus **works only if each dimensions are orthogonal to each other**.
- `ellipsoid.volume`: calculates the ellipsoid volume of a matrix. This function tries to determine the nature of the input matrix and uses one of these following methods to calculate the volume. You can always specify the method using `method = "my_choice"` to overrun the automatic method choice.

  - "eigen": this method directly calculates the eigen values from the input matrix (using eigen). This method is automatically selected if the input matrix is "distance like" (i.e. square with two mirrored triangles and a diagonal).
  - "pca": this method calculates the eigen values as the sum of the variances of the matrix (abs(apply(var(matrix),2, sum))). This is automatically selected if the input matrix is NOT "distance like". Note that this method is faster than "eigen" but only works if the input matrix is an ordinated matrix from a PCA, PCO, PCoA, NMDS or MDS.
  - "axes": this method calculates the actual semi axes length using the input matrix. It is never automatically selected. By default this method calculates the length of the major axes based on the 0.95 confidence interval ellipse but this can be modified by providing additional arguments from axis.covar.
  - <a numeric vector>: finally, you can directly provide a numeric vector of eigen values. This method is never automatically selected and overrides any other options.

- func.div: The functional divergence (Villeger et al. 2008): the ratio of deviation from the centroid (this is similar to FD::dbFD()$FDiv).
- func.eve: The functional evenness (Villeger et al. 2008): the minimal spanning tree distances evenness (this is similar to FD::dbFD()$FEve). If the matrix used is not a distance matrix, the distance method can be passed using, for example method = "euclidean" (default).
- mode.val: calculates the modal value of a vector.
- n.ball.volume: calculate the volume of the minimum n-ball (if sphere = TRUE) or of the ellipsoid (if sphere = FALSE).
- roundness: calculate the roundness of an elliptical representation of a variance-covariance matrix as the integral of the ranked distribution of the major axes. A value of 1 indicates a sphere, a value between 1 and 0.5 indicates a more pancake like representation and a value between 0.5 and 0 a more cigar like representation. You can force the variance-covariance calculation by using the option vcv = TRUE (default) that will calculate the variance-covariance matrix if the input is not one.

See also mean, median, sum or prod for commonly used summary metrics.

The currently implemented dimension-level 2 metrics are:

- ancestral.dist: calculates the distance between each elements coordinates in the matrix and their ancestors' coordinates (if to.root = FALSE; default) or to the root coordinates (if to.root = TRUE) for a given tree. The distance is calculate as Euclidean by default but can be changed through the methods argument (method = "euclidean"; default). Note that the matrix must contain data for both tips and nodes in the tree, otherwise you must provide a matrix to the argument reference.data that contains them. Note that if the function is used in dispRity, both the tree and reference.data can be automatically recycled from the dispRity object (if present).
- angles: calculates the angles of the main axis of variation per dimension in a matrix. The angles are calculated using the least square algorithm from the lm function. The unit of the angle can be changed through the unit argument (either "degree" (default), radian or slope) and a base angle to measure the angle from can be passed through the base argument (by default base = 0, measuring the angle from the horizontal line (note that the base argument has to be passed in the same unit as unit). When estimating the slope through lm, you can use the option significant to only consider significant slopes (TRUE) or not (FALSE - default).

- `centroids`: calculates the distance between each row and the centroid of the matrix (Laliberte 2010). This function can take an optional arguments `centroid` for defining the centroid (if missing (default), the centroid of the matrix is used). This argument can be either a subset of coordinates matching the matrix's dimensions (e.g. `c(0, 1, 2)` for a matrix with three columns) or a single value to be the coordinates of the centroid (e.g. `centroid = 0` will set the centroid coordinates to `c(0, 0, 0)` for a three dimensional matrix). NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `count.neighbours`: counts the number of other elements neigbhouring each element within a certain radius. This function can take the optional arguments `radius` that is the radius for counting the neighbours. This can be either missing (by default this is half the longest distance), a function to calculate the distance taking `x` as the sole argument (e.g. `sd` or `function(x) sum(x, na.rm = TRUE)/length(x)`) or a `numeric` or `integer` value. The other option is `relative` to make the counts relative to the number of elements (`relative = TRUE`; default) or not (`relative = FALSE`). NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `deviations`: calculates the minimal Euclidean distance between each element in and the hyperplane (or line if 2D, or a plane if 3D). You can specify equation of hyperplane of d dimensions in the $intercept + ax + by + ... + nd = 0$ format. For example the line $y = 3x + 1$ should be entered as `c(1, 3, -1)` or the plane $x + 2y - 3z = 44$ as `c(44, 1, 2, -3)`. If missing the `hyperplane` (default) is calculated using a least square regression using a gaussian `glm`. Extra arguments can be passed to `glm` through `....` When estimating the hyperplane, you can use the option `significant` to only consider significant slopes (`TRUE`) or not (`FALSE` - default).
- `displacements`: calculates the ratio between the distance to the centroid (see `centroids` above) and the distance from a reference (by default the origin of the space). The reference can be changed through the `reference` argument. NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `edge.length.tree`: calculates the edge length from a given tree for each elements present in the matrix. Each edge length is either measured between the element and the root of the tree (`to.root = TRUE` ; default) or between the element and its last ancestor (`to.root = FALSE`))
- `neighbours`: calculates the distance to a neighbour (Foote 1990). By default this is the distance to the nearest neighbour (`which = min`) but can be set to any dimension level - 1 function (e.g. `which = mean` gives the distance to the most average neighbour). NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `pairwise.dist`: calculates the pairwise distance between elements - calls `vegdist(matrix, method = method, diag = FALSE, upper = FALSE, ...)`. The distance type can be changed via the `method` argument (see `vegdist` - default: `method = "euclidean"`). This function outputs a vector of pairwise comparisons in the following order: d(A,B), d(A,C), d(B,C) for three elements A, B and C. NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `projections`: projects each element on a vector defined as (`point1`, `point2`) and measures some aspect of this projection. The different aspects that can be measured are:
  - `measure = "position"` (default), the distance of each element *on* the vector (`point1`, `point2`). Negative values means the element projects on the opposite direction of the vector (`point1`, `point2`).
  - `measure = "distance"`, the euclidean distance of each element *from* the vector (`point1`, `point2`).

- `measure = "degree"`, the angle between the vector (`point1`, `point2`) and any vector (`point1`, `element`) in degrees.
- `measure = "radian"`, the angle between the vector (`point1`, `point2`) and any vector (`point1`, `element`) in radians.
- `measure = "orthogonality"`, the angle between the vector (`point1`, `point2`) and any vector (`point1`, `element`) expressed in right angle ranging between 0 (non angle) and 1 (right angle).

By default, `point1` is the centre of the space (coordinates `0, 0, 0, ...`) and `point2` is the centroid of the space (coordinates `colMeans(matrix)`). Coordinates for `point1` and `point2` can be given as a single value to be repeated (e.g. `point1 = 1` is translated into `point1 = c(1, 1, ...)`) or a specific set of coordinates. Furthermore, by default, the space is scaled so that the vector (`point1`, `point2`) becomes the unit vector (distance (`point1`, `point2`) is set to 1; option `scale = TRUE`; default). You can use the unit vector of the space using the option `scale = FALSE`. Other options include the centering of the projections on 0.5 (`centre = TRUE`; default is set to `FALSE`) ranging the projection onto the vector (`point1`, `point2`) between -1 and 1 (higher or lower values project beyond the vector); and whether to output the projection values as absolute values (`abs = FALSE`; default is set to `FALSE`). These two last options only affect the results from `measure = "position"`.

- `projections.tree`: calculates the `projections` metric but drawing the vectors from a phylogenetic tree. This metric can intake any argument from `projections` (see above) but for `point1` and `point2` that are replaced by the argument `type`. `type` is a `vector` or a `list` of two elements that designates which vector to draw and can be any pair of the following options (the first element being the origin of the vector and the second where the vector points to):
  - `"root"`: the root of the tree (the first element in tree$node.label);
  - `"ancestor"`: the element's most recent ancestor;
  - `"tips"`: the centroid of the tips;
  - `"nodes"`: the centroid of all the nodes;
  - `"livings"`: the centroid of the tips the furthest from the root;
  - `"fossils"`: the centroid of all the tips that are not the furthest from the root;
  - any numeric values that can be interpreted as `point1` and `point2` in `projections`;
  - or a user defined function that with the inputs `matrix` and `tree` and `row` (the element's ID, i.e. the row number in `matrix`).

  NOTE: the elements to calculate the origin and end points of the vector are calculated by default on the provided input `matrix` which can be missing data from the tree if used with `custom.subsets` or `chrono.subsets`. You can always provide the full matrix using the option `reference.data = my_matrix`. Additional arguments include any arguments to be passed to `projections` (e.g. `centre` or `abs`).
- `quantiles`: calculates the quantile range of each axis of the matrix. The quantile can be changed using the `quantile` argument (default is `quantile = 95`, i.e. calculating the range on each axis that includes 95% of the data). An optional argument, `k.root`, can be set to `TRUE` to scale the ranges by using its kth root (where k are the number of dimensions). By default, `k.root = FALSE`.
- `radius`: calculates a distance from the centre of each axis. The `type` argument is the function to select which distance to calculate. By default `type = max` calculates the maximum distance between the elements and the centre for each axis (i.e. the radius for each dimensions)

- `ranges`: calculates the range of each axis of the matrix (Wills 2001). An optional argument, `k.root`, can be set to `TRUE` to scale the ranges by using its kth root (where k are the number of dimensions). By default, `k.root = FALSE`.
- `variances`: calculates the variance of each axis of the matrix (Wills 2001). This function can also take the `k.root` optional argument described above.
- `span.tree.length`: calculates the length of the minimum spanning tree (see `spantree`). This function can get slow with big matrices. To speed it up, one can directly use distance matrices as the multidimensional space.

The currently implemented between.groups metrics are:

- `disalignment`: calculates the rejection of a point from `matrix` from the major axis of `matrix2`. Options are, `axis` to choose which major axis to reject from (default is `axis = 1`); `level` for the ellipse' confidence interval (to calculate the axis) (default is `level = 0.95`) and `point.to.reject`, a numeric value for designating which point in `matrix` to use or a function for calculating it (default is `point.to.reject = colMeans` for `matrix`'s centroid).
- `group.dist`: calculates the distance between two groups (by default, this is the minimum euclidean vector norm distance between groups). Negative distances are considered as 0. This function must intake two matrices (`matrix` and `matrix2`) and the quantiles to consider. For the minimum distance between two groups, the 100th quantiles are considered (default: `probs = c(0,1)`) but this can be changed to any values (e.g. distance between the two groups accounting based on the 95th CI: `probs = c(0.025, 0.975)`; distance between centroids: `probs = c(0.5)`, etc...). This function is the linear algebra equivalent of the `hypervolume::hypervolume_distance` function.
- `point.dist`: calculates the distance between `matrix` and a point calculated from `matrix2`. By default, this point is the centroid of `matrix2`. This can be changed by passing a function to be applied to `matrix2` through the `point` argument (for example, for the centroid: `point.dist(..., point = colMeans)`). NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.
- `projections.between`: calculates the projection of the major axis between two matrices. It allows the same arguments as `projections`. This function measures the major axis from both input matrices, centre their origins and projects the end of the vector of `matrix` onto the vector from `matrix2`. Which axis to measure can be changed with the option `axis` (for the major axis, `axis = 1`; default) and the confidence interval can be changed using `level` (for the 95 confidence interval, `level = 0.95`; default - see `axis.covar` for more details).

When used in the `dispRity` function, optional arguments are declared after the `metric` argument: for example `dispRity(data, metric = centroids, centroid = 0, method = "manhattan")`

Author(s)

Thomas Guillerme

References

Donohue I, Petchey OL, Montoya JM, Jackson AL, McNally L, Viana M, Healy K, Lurgi M, O'Connor NE, Emmerson MC. 2013. On the dimensionality of ecological stability. Ecology letters. 16(4):421-9.

Lalibert'e E, Legendre P. 2010. A distance-based framework for measuring functional diversity from multiple traits. Ecology, 91(1), pp.299-305.

Vill'eger S, Mason NW, Mouillot D. 2008. New multidimensional functional diversity indices for a multifaceted framework in functional ecology. Ecology. 89(8):2290-301.

Wills MA. 2001. Morphological disparity: a primer. In Fossils, phylogeny, and form (pp. 55-144). Springer, Boston, MA.

Foote, M. 1990. Nearest-neighbor analysis of trilobite morphospace. Systematic Zoology, 39(4), pp.371-382.

Guillerme T, Puttick MN, Marcy AE, Weisbecker V. 2020. Shifting spaces: Which disparity or dissimilarity measurement best summarize occupancy in multidimensional spaces?. Ecology and evolution. 10(14):7261-75.

See Also

dispRity and make.metric.

Examples

```
## A random matrix
dummy_matrix <- matrix(rnorm(90), 9, 10)

## ancestral.dist
## A random tree with node labels
rand_tree <- rtree(5) ; rand_tree$node.label <- paste0("n", 1:4)
## Adding the tip and node names to the matris
rownames(dummy_matrix) <- c(rand_tree$tip.label, rand_tree$node.label)
## Calculating the distances to the ancestors
ancestral.dist(dummy_matrix, tree = rand_tree)
## Calculating the manhattan distances to the root
ancestral.dist(dummy_matrix, tree = rand_tree,
               to.root = TRUE, method = "manhattan")

## angles
## The angles in degrees of each axis
angles(dummy_matrix)
## The angles in slope from the 1:1 slope (Beta = 1)
angles(dummy_matrix, unit = "slope", base = 1)

## centroids
## Distances between each row and centroid of the matrix
centroids(dummy_matrix)
## Distances between each row and an arbitrary point
centroids(dummy_matrix, centroid = c(1,2,3,4,5,6,7,8,9,10))
## Distances between each row and the origin
centroids(dummy_matrix, centroid = 0)

## convhull.surface
## Making a matrix with more elements than dimensions (for convhull)
thinner_matrix <- matrix(rnorm(90), 18, 5)
## Convex hull hypersurface of a matrix
```

```
convhull.surface(thinner_matrix)

## convhull.volume
## Convex hull volume of a matrix
convhull.volume(thinner_matrix)

## count.neigbhours
## Counting the number of neighbours within a radius of half the traitspace
count.neighbours(dummy_matrix)
## The absolute number of neighbours within a radius of 3
count.neighbours(dummy_matrix, radius = 3, relative = FALSE)
## The relative number of neighbours within a radius of one standard deviation
count.neighbours(dummy_matrix, radius = sd, relative = FALSE)

## deviations
## The deviations from the least square hyperplane
deviations(dummy_matrix)
## The deviations from the plane between the x and y axis
deviations(dummy_matrix, hyperplane = c(0,1,1,0,0,0,0,0,0,0,0))

## diagonal
## Matrix diagonal
diagonal(dummy_matrix) # WARNING: only valid if the dimensions are orthogonal

## disalignment
## Two dummy matrices
matrix_1 <- matrix(rnorm(16), 4, 4)
matrix_2 <- matrix(rnorm(16), 4, 4)
## Measuring the disalignment of matrix_1 from matrix_2
disalignment(matrix_1, matrix_2)
## Same but using the 2nd major axis of the 0.75 CI ellipse
## from matrix_2 and the first point from matrix_1.
disalignment(matrix_1, matrix_2,
             axis = 2, level = 0.75,
             point.to.reject = 1)

## displacements
## displacement ratios (from the centre)
displacements(dummy_matrix)
## displacement ratios (from an arbitrary point)
displacements(dummy_matrix, reference = c(1,2,3,4,5,6,7,8,9,10))
## displacement ratios from the centre (manhattan distance)
displacements(dummy_matrix, method = "manhattan")

## edge.length.tree
## Making a dummy tree with node labels
dummy_tree <- makeNodeLabel(rtree((nrow(dummy_matrix)/2)+1))
## Naming the elements in the matrix
named_matrix <- dummy_matrix
rownames(named_matrix) <- c(dummy_tree$tip.label,
                            dummy_tree$node.label)
## The total edge length of each element in the matrix (to the root)
edge.length.tree(named_matrix, tree = dummy_tree)
```

```
## The edge lengths for each edge leading to the elements in the matrix
edge.length.tree(named_matrix, tree = dummy_tree, to.root = FALSE)

## ellipsoid.volume
## Ellipsoid volume of a matrix
ellipsoid.volume(dummy_matrix)
## Calculating the same volume with provided eigen values
ordination <- prcomp(dummy_matrix)
## Calculating the ellipsoid volume by providing your own eigen values
ellipsoid.volume(ordination$x, method = ordination$sdev^2)

## func.div
## Functional divergence
func.div(dummy_matrix)

## func.eve
## Functional evenness
func.eve(dummy_matrix)
## Functional evenness (based on manhattan distances)
func.eve(dummy_matrix, method = "manhattan")

## group.dist
## The distance between groups
dummy_matrix2 <- matrix(runif(40, min = 2, max = 4), 4, 10)
## The minimum distance between both groups
group.dist(dummy_matrix, dummy_matrix2)
## The distance between both groups' centroids
group.dist(dummy_matrix, dummy_matrix2, probs = 0.5)
## The minimum distance between the 50% CI of each group
group.dist(dummy_matrix, dummy_matrix2, probs = c(0.25, 0.75))

## mode.val
## Modal value of a vector
mode.val(dummy_matrix)

## neighbours
## The nearest neighbour euclidean distances
neighbours(dummy_matrix)
## The furthest neighbour manhattan distances
neighbours(dummy_matrix, which = max, method = "manhattan")

## pairwise.dist
## The pairwise distance
pairwise.dist(dummy_matrix)
## The average squared pairwise distance
mean(pairwise.dist(dummy_matrix)^2)
## equal to:
# geiger::disparity(data = dummy_matrix)

## point.dist
## The distances from the rows dummy_matrix
## to the centroids of dummy_matrix2
```

```
point.dist(dummy_matrix, dummy_matrix2)
## The average distances from dummy_matrix
## to the centroids of dummy_matrix2
mean(point.dist(dummy_matrix, dummy_matrix2))
## The manhattan distance from the rows dummy_matrix
## to the standard deviation of dummy_matrix2
point.dist(dummy_matrix, dummy_matrix2, point = sd, method = "manhattan")

## projections
## The distances on the vector defined from the centre of
## the matrix to its centroid (default)
projections(dummy_matrix)
## The distances from the vector defined from the third
## element of the matrix to the point of coordinated
## c(1,1,1, ...) the matrix to its centroid (default)
projections(dummy_matrix, measure = "distance",
            point1 = dummy_matrix[3, ],
            point2 = 1)

## projections.tree
## Making a dummy tree with node labels
dummy_tree <- makeNodeLabel(rtree((nrow(dummy_matrix)/2)+1))
## Naming the elements in the matrix
named_matrix <- dummy_matrix
rownames(named_matrix) <- c(dummy_tree$tip.label,
                            dummy_tree$node.label)
## The projection on the vector defined from the root of
## the tree to the ancestor of each element in the matrix
projections.tree(named_matrix, dummy_tree,
                 type = c("root", "ancestor"))
## The rejection from the vector defined from the centroid
## of the nodes to the centroids of the tips
projections.tree(named_matrix, dummy_tree,
                 type = c("nodes", "tips"),
                 measure = "distance")
## A user function that define coordinates based on the
## centroid of the three first nodes
user.fun <- function(matrix, tree, row = NULL) {
     return(colMeans(matrix[tree$node.label[1:3], ]))
}
## The projection on the vector defined by the coordinates
## 0,0,0 and a user defined function
projections.tree(named_matrix, dummy_tree,
                 type = c(0, user.fun))

## projections.between
## Two dummy matrices
matrix_1 <- matrix(rnorm(16), 4, 4)
matrix_2 <- matrix(rnorm(16), 4, 4)
## Projecting the major axis of matrix_2 onto the one from matrix_1
projections.between(matrix_1, matrix_2)
## Projecting both second major 0.75 axes
## and getting the rejections (see projections() for option details)
```

```
projections.between(matrix_1, matrix_2,
                    measure = "distance",
                    axis = 2, level = 0.75)

## quantiles
## The 95 quantiles
quantiles(dummy_matrix)
## The 100 quantiles (which are equal to the ranges)
quantiles(dummy_matrix, quantile = 100) == ranges(dummy_matrix) # All TRUE

## radius
## The maximal radius of each axis (maximum distance from centre of each axis)
radius(dummy_matrix)

## ranges
## ranges of each column in a matrix
ranges(dummy_matrix)
## ranges of each column in the matrix corrected using the kth root
ranges(dummy_matrix, k.root = TRUE)

## roundness
## calculating the variance-covariance of the dummy_matrix
vcv <- var(dummy_matrix)
## calculating the roundness of it
roundness(vcv)
## calculating the roundness of the dummy matrix by calculating the vcv
roundness(dummy_matrix, vcv = TRUE)

## span.tree.length
## Minimum spanning tree length (default)
span.tree.length(dummy_matrix)
## Minimum spanning tree length from a distance matrix (faster)
distance <- as.matrix(dist(dummy_matrix))
span.tree.length(distance)
## Minimum spanning tree length based on Manhattan distance
span.tree.length(dummy_matrix, method = "manhattan")
span.tree.length(as.matrix(dist(dummy_matrix, method = "manhattan"))) # Same

## variances
## variances of a each column in the matrix
variances(dummy_matrix)
## variances of a each column in the matrix corrected using the kth root
variances(dummy_matrix, k.root = TRUE)
```

`dispRity.per.group` *Disparity in different groups.*

Description

Performs a disparity analysis between groups.

Usage

```
dispRity.per.group(data, group, metric = c(median, centroids), ...)
```

Arguments

data	An ordinated matrix.
group	A list of row numbers for each group.
metric	A vector containing one to three functions (default = c(median, centroids)) (see dispRity for details).
...	Optional arguments to be passed to custom.subsets, boot.matrix and dispRity.

Details

Note that this is a wrapper function that allows users to run a basic disparity among groups analysis without too much effort. As such it has a lot of defaults described in the functions that make up the analysis. See custom.subsets, boot.matrix, dispRity.metric, summary.dispRity, plot.dispRity for more details of the defaults used in each of these functions. Note that any of these defaults can be changed within the disparity.through.time function.

Value

A dispRity object that can be passed to summary or plot.

Author(s)

Thomas Guillerme

See Also

custom.subsets, boot.matrix, dispRity.metric, summary.dispRity, plot.dispRity.

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_mat50)

## Run a simple disparity per group analysis comparing stem and crown mammals
result <- dispRity.per.group(BeckLee_mat50, list(crown = c(16, 19:41, 45:50),
                             stem = c(1:15, 17:18, 42:44)))
summary(result) ; plot(result)

## This is equivalent to run the following decomposed code
dispRity(boot.matrix(custom.subsets(BeckLee_mat50, list(crown = c(16, 19:41, 45:50),
                                                        stem = c(1:15, 17:18, 42:44))),
                     bootstraps = 100),
         metric = c(median, centroids))
```

`dispRity.through.time` *Disparity through time.*

Description

Performs a disparity through time analysis.

Usage

```
dispRity.through.time(data, tree, time, metric = c(median, centroids), ...)
```

Arguments

`data`	An ordinated `matrix`.
`tree`	A `phylo` object.
`time`	A `numeric` value for the number of subsets to create.
`metric`	A vector containing one to three functions (default = `c(median, centroids)`) (see `dispRity` for details).
`...`	Optional arguments to be passed to `chrono.subsets`, `boot.matrix` and `dispRity`.

Details

By default the time subsets use `method = "discrete"`, the matrix is bootstrapped 100 times.

Note that this is a wrapper function that allows users to run a basic disparity-through-time analysis without too much effort. As such it has a lot of defaults described in the functions that make up the analysis. See `chrono.subsets`, `boot.matrix`, `dispRity.metric`, `summary.dispRity`, `plot.dispRity` for more details of the defaults used in each of these functions. Note that any of these defaults can be changed within the `disparity.through.time` function.

Value

A `dispRity` object that can be passed to `summary` or `plot`.

Author(s)

Thomas Guillerme

See Also

`chrono.subsets`, `boot.matrix`, `dispRity.metric`, `summary.dispRity`, `plot.dispRity`.

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_mat50) ; data(BeckLee_tree)

## Run a simple disparity through time analysis (with three time bins)
result <- dispRity.through.time(BeckLee_mat50, BeckLee_tree, 3)
summary(result) ; plot(result)

## This is equivalent to run the following decomposed code
dispRity(boot.matrix(chrono.subsets(BeckLee_mat50, BeckLee_tree, time = 3, method = "discrete"),
                     bootstraps = 100),
         metric = c(median, centroids))
```

---

distance.randtest *Randtest distance*

Description

Measures the distance between the observed statistic from a `"randtest"` object and some specific quantile of the simulated data.

Usage

```
distance.randtest(xtest, quantile = c(0.025, 0.975), abs = FALSE)
```

Arguments

`xtest`	an object of class `"randtest"`
`quantile`	a `numeric` value for the quantile edges to compare the observed data to on either sides (by default `quantile = c(0.025. 0.975)`).
`abs`	`logical`, whether to calculate the distance as an absolute value (`TRUE`) or not (`FALSE` - default).

Details

To compare the observed value to the simulated median value, you can use `quantile = 0.5`. Also note that when using `abs = FALSE` (default), a negative value means that the observed statistic is within the request quantiles.

Author(s)

Thomas Guillerme

See Also

`randtest` `randtest.dispRity`

Examples

```
## Simple example
dummy_matrix <- matrix(rnorm(500), 100, 5)

## Testing whether the mean of a random subset
## is different than the means of 100 subsets
dummy_test <- randtest.dispRity(dummy_matrix,
                                subset = sample(1:100, 20),
                                metric = mean)
dummy_test ; plot(dummy_test)

## The distance between the observed data and the 95% quantile
distance.randtest(dummy_test)

## The absolute distance from the median
distance.randtest(dummy_test, quantile = 0.5, abs = TRUE)
```

`dtt.dispRity` *dtt dispRity (from* `geiger::dtt`*)*

Description

A wrapper for the `geiger::dtt` function working with any disparity metric.

Usage

```
dtt.dispRity(
  data,
  metric,
  tree,
  nsim = 0,
  model = "BM",
  alternative = "two-sided",
  scale.time = TRUE,
  ...
)
```

Arguments

`data`	A `dispRity` object or a `matrix`
`metric`	The disparity metric to be passed to `dispRity`.
`tree`	A `phylo` object matching the data and with a `root.time` element. Can be missing if `data` has a `tree` component.
`nsim`	The number of simulations to calculate null disparity-through-time.
`model`	A evolutionary model for the simulations (see `geiger::sim.char` - default is `"BM"`).

alternative	The H1 alternative (for calculating the p-value). Can be "two-sided" (default), "greater" or "lesser"; see details.
scale.time	Optional, whether to scale the time (between 0 and 1; TRUE, default) or not (FALSE).
...	Any other arguments to be passed to geiger::dtt.

Details

See geiger::dtt for details.

Author(s)

Thomas Guillerme

See Also

test.dispRity, custom.subsets, chrono.subsets, plot.dispRity.

Examples

```
## Loading morphological data and a tree
data(BeckLee_mat50)
data(BeckLee_tree)

## The average squared pairwise distance metric (used in geiger::dtt)
average.sq <- function(X) mean(pairwise.dist(X)^2)

## Calculate the disparity of the dataset using dtt.dispRity
dispRity_dtt <- dtt.dispRity(data = BeckLee_mat50, metric = average.sq,
                             tree = BeckLee_tree, nsim = 20)

## Plotting the results
plot(dispRity_dtt)
```

---

extinction.subsets *Getting the time subsets before and after an extinction event*

Description

Getting the reference (pre-extinction) and the comparison (post-extinction) time subsets

Usage

```
extinction.subsets(data, extinction, lag = 1, names = FALSE, as.list = FALSE)
```

Arguments

data	a dispRity object.
extinction	numerical, the time at the extinction event.
lag	numerical, the lag effect (i.e. how many subsets after the extinction to consider - default = 1).
names	logical, whether to display the bins names (TRUE) or not (FALSE - default).
as.list	logical, whether to output the results as a list for test.dispRity (TRUE) or not (FALSE - default).

Author(s)

Thomas Guillerme

See Also

chrono.subsets, test.dispRity

Examples

```
## Loading some disparity data
data(disparity)

## Time subsets for the K-Pg extinction (66 Mya)
extinction.subsets(disparity, 66, names = TRUE)

## Extinction with a lag effect of 3 slices
extinction_time <- extinction.subsets(disparity, 66, lag = 3, as.list = TRUE)

## Testing the extinction effect with a lag
test.dispRity(disparity, wilcox.test, comparisons = extinction_time,
              correction = "bonferroni")
```

---

geomorph.ordination *Imports data from geomorph*

Description

Takes geomorph Procrustes object or a geomorph.data.frame object and ordinates it.

Usage

```
geomorph.ordination(data, ordinate = TRUE, ...)
```

Arguments

data	An array (p x k x n) typically obtained from a Procrustes superimposition geomorph::gpagen or a geomorph::geomorph.data.frame object.
ordinate	Logical, whether to ordinate the data using prcomp (TRUE; default) or not (FALSE; the code then returns the raw coordinates matrix).
...	Any optional arguments to be passed to prcomp (is ignored if ordinate = FALSE).

Details

If data is a geomorph.data.frame object containing factors, directly performs a custom.subsets using these factors.

Value

A matrix or a dispRity object.

See Also

geomorph::gpagen, geomorph::morphol.disparity, prcomp, custom.subsets, chrono.subsets, boot.matrix, dispRity.

Examples

```
## Not run:
require(geomorph)
## Loading the plethodon dataset
data(plethodon)

## Performing a Procrustes transform
procrustes <- geomorph::gpagen(plethodon$land, PrinAxes = FALSE)

## Obtaining the ordination matrix
geomorph.ordination(procrustes)

## Using a geomorph.data.frame
geomorph_df <- geomorph.data.frame(procrustes, species = plethodon$species)

geomorph.ordination(geomorph_df)

## Calculating disparity from dispRity or geomorph::morphol.disparity
geomorph_disparity <- geomorph::morphol.disparity(coords ~ 1,
                      groups= ~ species, data = geomorph_df)
dispRity_disparity <- dispRity(geomorph.ordination(geomorph_df),
                      metric = function(X) return(sum(X^2)/nrow(X)))

## Extracting the raw disparity values
geomorph_val <- round(as.numeric(geomorph_disparity$Procrustes.var), 15)
dispRity_val <- as.vector(summary(dispRity_disparity, digits = 15)$obs)

## Comparing the values (to the 15th decimal!)
geomorph_val == dispRity_val # all TRUE
```

```
## End(Not run)
```

`get.bin.ages` *Get time bins ages*

Description

Gets time bins for a specific tree using stratigraphy

Usage

```
get.bin.ages(tree, what = "End", type = "Age", ICS = 2015)
```

Arguments

`tree`	A `phylo` object with a `$root.time` component
`what`	Which data to output. Can be `"Start"`, `"End"` (default), `"Range"` or `"Midpoint"`.
`type`	The type of stratigraphic frame. Can be `"Age"` (default), `"Eon"`, `"Epoch"`, `"Era"` or `"Period"`.
`ICS`	The reference year of the International Commission on Stratigraphy (default = `2015`).

Author(s)

Thomas Guillerme

See Also

`chrono.subsets`

Examples

```
## Loading the data
data(BeckLee_tree)
data(BeckLee_mat50)

## Getting the stratigraphic data
stratigraphy <- get.bin.ages(BeckLee_tree)

## Making stratigraphic time subsets
chrono.subsets(BeckLee_mat50, tree = BeckLee_tree, method = "discrete",
               time = stratigraphy)
```

`get.contrast.matrix` *Generates a contrast matrix.*

Description

Creates a contrast matrix using the observed character states in an input matrix.

Usage

```
get.contrast.matrix(matrix)
```

Arguments

`matrix`	a discrete morphological character matrix.

Author(s)

Thomas Guillerme

See Also

`check.morpho`

Examples

```
## A random multistate matrix
random_matrix <- matrix(sample(c(0,1,2), 100, TRUE), 10, 10)

## Get the contrast matrix
get.contrast.matrix(random_matrix)

## Adding inapplicable and missing data to the matrix
random_matrix[sample(1:100, 10)] <- "?"
random_matrix[sample(1:100, 10)] <- "-"

## Get the contrast matrix
get.contrast.matrix(random_matrix)
```

`get.matrix` *Extract elements from a* `dispRity` *object.*

Description

Extract a matrix or the disparity results from a `dispRity`.

Usage

```
get.matrix(data, subsets, rarefaction, bootstrap, matrix)

get.disparity(data, subsets, rarefaction, observed, concatenate)
```

Arguments

`data`	A `dispRity` object.
`subsets`	Optional, a `numeric` or `character` for which subsets to get (if missing, the value for all subsets are given).
`rarefaction`	Optional, a single `numeric` value corresponding to the rarefaction level (as the number of elements; if missing, the non-rarefied values are output).
`bootstrap`	Optional, a `numeric` value to select a specific bootstrap draw (`0` is no bootstrap).
`matrix`	A `numeric` value of which matrix to select (default is `1`).
`observed`	A `logical` value indicating whether to output the observed (`TRUE` (default)) or the bootstrapped values (`FALSE`).
`concatenate`	When the disparity metric is a distribution, whether to concatenate it returning the median (`TRUE`; default) or to return each individual values.

Author(s)

Thomas Guillerme

See Also

`dispRity`, `get.subsets`.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## To get the original matrix
get.matrix(disparity)

## To get the un-bootstrapped matrix from the subset called "80"
get.matrix(disparity, subsets = "80")
```

```
## To get the 52nd bootstrap draw of the second rarefaction level (15) of the
## same subset
get.matrix(disparity, subsets = 2, rarefaction = 2, bootstrap = 52)

## Extracting the observed disparity
get.disparity(disparity)

## Extracting the bootstrapped disparity
boot_disp <- get.disparity(disparity, observed = FALSE)
str(boot_disp)
## Or only the rarefied (5) data
boot_disp_rare <- get.disparity(disparity, observed = FALSE,
     rarefaction = 5)
```

`get.subsets` *Extracts or modify subsets from a* `dispRity` *object.*

Description

Extracting or modify some subsets' data and information from a `dispRity` object.

Usage

```
n.subsets(data)

name.subsets(data)

size.subsets(data)

get.subsets(data, subsets)

combine.subsets(data, subsets)
```

Arguments

`data`	A `dispRity` object.
`subsets`	Either a `vector` of the number or name of the subsets to merge or a single. But see details for `combine.subsets`.

Details

For the function `combine.subsets`, the argument `subsets` can ALSO be a `numeric` value of the minimum of elements for each series. If `subset` is a vector, the subsets are merged in the given input order. `c(1, 3, 4)` will merge subsets 1 and 3 into 4, while the opposite, `c(3, 4, 1)` will merge subsets 3 and 4 into 1. When a single numeric value is given, subsets are merged with the next subset until the correct number of elements for each subset is reached (apart from the last subset that gets merged with the previous one).

Author(s)

Thomas Guillerme

See Also

dispRity, get.disparity.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## How many subsets are in disparity?
n.subsets(disparity)

## What are the subset names
name.subsets(disparity)

## What are the number of elements per subsets?
size.subsets(disparity)

## Get one subset
get.subsets(disparity, "60")

## Get two subsets
get.subsets(disparity, c(1,5))

## Generate subsets from a dummy matrix
dummy_matrix <- matrix(rnorm(120), 40, dimnames = list(c(1:40)))
dummy_subsets <- custom.subsets(dummy_matrix,
     group = list("a" = c(1:5), "b" = c(6:10), "c" = c(11:20),
                  "d" = c(21:24), "e" = c(25:30), "f" = c(31:40)))

## Merging the two first subsets
combine.subsets(dummy_subsets, c(1,2))

## Merging the three subsets by name
combine.subsets(dummy_subsets, c("d", "c", "e"))

## Merging the subsets to contain at least 20 taxa
combine.subsets(dummy_subsets, 10)
```

---

make.dispRity *Make and fill* dispRity.

Description

Creating an empty dispRity object from a matrix

Usage

```
make.dispRity(data, tree, call, subsets)

fill.dispRity(data, tree, check)

remove.dispRity(data, what)
```

Arguments

data	A matrix.
tree	Optional, a phylo or multiPhylo object.
call	Optional, a list to be a dispRity call.
subsets	Optional, a list to be a dispRity subsets list.
check	Logical, whether to check the data (TRUE; default, highly advised) or not (FALSE).
what	Which elements to remove. Can be any of the following: "subsets", "bootstraps", "covar", "tree", "disparity". See details.

Details

When using remove.dispRity, the function recursively removes any other data depending on "what". For example, for a data with disparity calculated for bootstrapped subsets, removing the subsets (what = "subsets") also removes the bootstraps and the disparity data. But removing the bootstraps (what = "bootstraps") removes only the bootstraps draws and the disparity relating to the bootstraps (but keeps the subsets and the non-bootstrapped disparity values).

Author(s)

Thomas Guillerme

Examples

```
## An empty dispRity object
make.dispRity()

## Still an empty dispRity object (with a matrix)
(empty <- make.dispRity(data = matrix(rnorm(12), ncol = 3)))

## A dispRity object with a matrix of 4*3
fill.dispRity(empty)

## A dispRity object with a tree
my_tree <- rtree(4, tip.label = c(1:4))
fill.dispRity(empty, tree = my_tree)
```

`make.metric` *Creating disparity metrics*

Description

Testing the dimension-level of disparity metrics

Usage

```
make.metric(
  fun,
  ...,
  silent = FALSE,
  check.between.groups = FALSE,
  data.dim,
  tree = NULL,
  covar = FALSE,
  get.help = FALSE
)
```

Arguments

`fun`	A `function`.
`...`	Some arguments to be passed to `fun`.
`silent`	`logical`; if `FALSE` (default), the function will be verbose and give no output; if `TRUE`, the function will only output the function's dimension-level.
`check.between.groups`	`logical`; if `TRUE`, the function will output a named list containing the metric level and a logical indicating whether the metric can be used between groups or not. If `FALSE` (default) the function only outputs the metric level.
`data.dim`	optional, two `numeric` values for the dimensions of the matrix to run the test function testing. If missing, a default 5 rows by 4 columns matrix is used.
`tree`	optional, a `phylo` object.
`covar`	`logical`, whether to treat the metric as applied the a `data$covar` component (`TRUE`) or not (`FALSE`; default).
`get.help`	`logical`, whether to also output the `dist.helper` if the metric has a `dist.help` argument (`TRUE`) or not (`FALSE`; default).

Details

This function tests:

- 1: if your function can deal with a matrix as an `input`.
- 2: which dimension-level is your function (1, 2 or 3, see `dispRity.metric`).
- 3: whether the function can properly be implemented in the `dispRity` function.

The three different metric levels correspond to the dimensions of the output and are:

- "dimension-level 1": for functions that decompose a `matrix` into a single value.
- "dimension-level 2": for functions that decompose a `matrix` into a `vector`.
- "dimension-level 3": for functions that transform the `matrix` into another `matrix`.

For example, the disparity metric `sum` of `variances` is composed of two metric dimension-levels:

- The `variances` (dimension-level 2) that calculates the variances for each column in a matrix (aggregates a `matrix` into a `vector`).
- The `sum` (dimension-level 1) that transforms the `vector` of variances into a single value.

See function example for a concrete illustration (three different dimension-levels of the function `sum`).

HINT: it is better practice to name the first argument of `fun` `matrix` to avoid potential argument conflicts down the line (the `dispRity` function assumes the `matrix` argument for the parsing the metrics).

The input `fun` can be a "normal" metric function (i.e. that takes a matrix as first argument) or a "between.groups" metric (i.e. that takes two matrix as arguments). If the arguments are named `matrix` and `matrix2`, the metric will be assumed to be "between.groups" and be run in a `for` loop rather than a `apply` loop in `dispRity`.

Author(s)

Thomas Guillerme

See Also

`dispRity`, `dispRity.metric`.

Examples

```
## A dimension-level 1 function
my_fun <- function(matrix) sum(matrix)
make.metric(my_fun)

## A dimension-level 2 function
my_fun <- function(matrix) apply(matrix, 2, sum)
make.metric(my_fun)

## A dimension-level 3 function
my_fun <- function(matrix) (matrix + sum(matrix))
make.metric(my_fun)
```

match.tip.edge *Match tips or nodes edge vector*

Description

Match a vector of tips or tips and nodes with the an edge list from a "phylo" or "multiPhylo".

Usage

```
match.tip.edge(
  vector,
  phylo,
  replace.na,
  use.parsimony = TRUE,
  to.root = FALSE
)
```

Arguments

vector	a vector of variables (equal to the number of tips or to the number of tips and nodes) or a vector of tips and nodes names or IDs.
phylo	a phylo or multiPhylo object.
replace.na	optional, what to replace NAs with.
use.parsimony	logical, whether to also colour internal edges parsimoniously (TRUE - default; i.e. if two nodes have the same unique ancestor node and the same variable, the ancestor node is assume to be the of the same value as its descendants) or not (FALSE).
to.root	logical, if vector is a list of tips and nodes, whether to colour internal edges all the way to the root (TRUE) or not (FALSE - default).

Value

If the input vector is a vector of variables, the function returns a vector of variables equal to the number of edges in the tree (or a list of vectors if the phylo input is of class "multiPhylo"). Else it returns an integer vector for the selected edges.

Author(s)

Thomas Guillerme

Examples

```
## A random tree
tree <- rtree(20)

## A random vector of two variables for each tips
tip_values <- sample(c("blue", "red"), 20, replace = TRUE)
```

```
## Matching the colors (blue and red) to the tips descendants
edge_colors <- match.tip.edge(tip_values, tree, replace.na = "grey")

## Plotting the results
plot(tree, show.tip.label = FALSE, edge.color = edge_colors)
tiplabels(1:20, bg = tip_values)

## Same but without assuming parsimony for the internal nodes
plot(tree, show.tip.label = FALSE,
     edge.color = match.tip.edge(tip_values, tree,
                                 use.parsimony = FALSE,
                                 replace.na = "grey"))

## Matching the tips and nodes colors with the edges
node_values <- sample(c("blue", "red"), 19, replace = TRUE)
edge_colors <- match.tip.edge(c(tip_values, node_values), tree)
plot(tree, show.tip.label = FALSE, edge.color = edge_colors)
tiplabels(1:20, bg = tip_values)
nodelabels(1:19, bg = node_values)

## Matching the tips and nodes colours to the root
data(bird.orders)

## Getting the bird orders starting with a "C"
some_orders <- sort(bird.orders$tip.label)[4:9]

## Get the edges linking these orders
edges_of_interest <- match.tip.edge(vector = some_orders,
                                    phylo  = bird.orders)

## Create a colour vector for all edges
all_edges <- rep("grey", Nedge(bird.orders))
## Replacing the edges of interest by another colour
all_edges[edges_of_interest] <- "black"

## Plot the results
plot(bird.orders, edge.color = all_edges)
```

`MCMCglmm.subsets` *MCMCglmm.subsets*

Description

Creating a dispRity object from a MCMCglmm posterior output

Usage

```
MCMCglmm.subsets(
```

```
  data,
  posteriors,
  group,
  tree,
  rename.groups,
  set.loc = TRUE,
  ...
)
```

Arguments

`data`	The `data.frame` or `matrix` used for the `MCMCglmm` model.
`posteriors`	A `MCMCglmm` object, the posteriors of the model.
`group`	Optional, a named vector of which group to include from the posteriors (if left empty the random and residual terms are used). See details.
`tree`	Optional, the tree(s) used in the MCMCglmm analyses.
`rename.groups`	Optional, a vector of group names for renaming them. See details.
`set.loc`	Optional, if no location is available for a subset (`$Sol = 0`), set the location of the subset from `data` (the centroid of the group) (default is `TRUE`).
`...`	Optional arguments to be passed to `MCMCglmm.covars`.

Details

- For the `group` option, the group names must be ones found in the `posteriors` formula in the format <*Type* = *Term:FactorLevel*> as returned by `MCMCglmm.levels(posteriors)`. For example, for returning two random effect, the phylogenetic one (`"animal"`) and one for a specific clade (say the 2nd clade) as well as two residual terms for a specific factor (say level 1 and 4) you can use `group = c(random = "animal", random = "animal:clade2", residual = "units:myfactor1", residual = "units:myfactor4")`.
- For the `rename.groups` option, the vector must be of class `"character"` and must of the same length as the number of random and residual terms in `posteriors` or of `group` argument (if used). If the `group` argument is left empty, the groups are extracted from the `posteriors` in the following order: the random terms first then the residual terms as specified in the `posteriors` object formulas (respectively `posteriors$Random$formula` and `posteriors$Residual$formula`).

NOTE that the output `dispRity` inherits the dimensions used in the `posteriors` argument. You can always check the selected dimensions using: `data$call$dimensions`

Author(s)

Thomas Guillerme

See Also

`dispRity` `covar.plot`

Examples

```
data(charadriiformes)

## Creating a dispRity object from the charadriiformes model
MCMCglmm.subsets(data       = charadriiformes$data,
                 posteriors = charadriiformes$posteriors)

## Same but selecting only the three first random terms
MCMCglmm.subsets(data       = charadriiformes$data,
                 posteriors = charadriiformes$posteriors,
                 tree       = charadriiformes$tree,
                 group      = MCMCglmm.levels(
                                  charadriiformes$posteriors)[1:3],
                 rename.groups = c("gulls", "plovers", "sandpipers"))
```

`MCMCglmm.utilities` *MCMCglmm object utility functions*

Description

Different utility functions to extract aspects of a `MCMCglmm` object.

Usage

```
MCMCglmm.traits(MCMCglmm)

MCMCglmm.levels(MCMCglmm, convert)

MCMCglmm.sample(MCMCglmm, n)

MCMCglmm.covars(MCMCglmm, n, sample)

MCMCglmm.variance(MCMCglmm, n, sample, levels, scale)
```

Arguments

`MCMCglmm`	A `MCMCglmm` object.
`convert`	Logical, whether to return the raw term names names as expressed in the model column names (`FALSE`) or to convert it to something more reader friendly (`TRUE`; default).
`n`	Optional, a number of random samples to extract.
`sample`	Optional, the specific samples to extract (is ignored if `n` is present).
`levels`	Optional, a vector `"character"` values (matching `MCMCglmm.levels(..., convert = TRUE)`) or of `"numeric"` values designating which levels to be used to calculate the variance (if left empty, all the levels are used).

scale	Logical, whether to scale the variance relative to all the levels (TRUE; default) or not (FALSE)/

Details

- MCMCglmm.levels returns the different random and residual terms levels of a MCMCglmm object. This function uses the default option convert = TRUE to convert the names into something more readable. Toggle to convert = FALSE for the raw names.
- MCMCglmm.traits returns the column names of the different traits of a MCMCglmm formula object.
- MCMCglmm.sample returns a vector of sample IDs present in the MCMCglmm object. If n is missing, all the samples IDs are returned. Else, a random series of sample IDs are returned (with replacement if n greater than the number of available samples).
- MCMCglmm.covars returns a list of covariance matrices and intercepts from a MCMCglmm object (respectively from MCMCglmm$VCV and MCMCglmm$Sol). By default, all the covariance matrices and intercepts are returned but you can use either of the arguments sample to return specific samples (e.g. MCMCglmm.covars(data, sample = c(1, 42)) for returning the first and 42nd samples) or n to return a specific number of random samples (e.g. MCMCglmm.covars(data, n = 42) for returning 42 random samples).
- MCMCglmm.variance returns a list of covariance matrices and intercepts from a MCMCglmm object (respectively from MCMCglmm$VCV and MCMCglmm$Sol). By default, all the covariance matrices and intercepts are returned but you can use either of the arguments sample to return specific samples (e.g. MCMCglmm.covars(data, sample = c(1, 42)) for returning the first and 42nd samples) or n to return a specific number of random samples (e.g. MCMCglmm.covars(data, n = 42) for returning 42 random samples).

Author(s)

Thomas Guillerme

See Also

MCMCglmm.subsets

Examples

```
## Loading the charadriiformes model
data(charadriiformes)
model <- charadriiformes$posteriors
class(model) # is MCMCglmm

## Get the list of levels from the model
MCMCglmm.levels(model)
## The raw levels names (as they appear in the MCMCglmm object)
MCMCglmm.levels(model, convert = FALSE)

## Get the traits names from the model
MCMCglmm.traits(model)
```

```
## Get all the available samples in the model
length(MCMCglmm.sample(model))
## Get 5 random sample IDs from the model
MCMCglmm.sample(model, n = 5)

## Get one specific samples from the model
MCMCglmm.covars(model, sample = 42)
## Get two random samples from the model
MCMCglmm.covars(model, n = 2)

## Get the variance for each terms in the model
terms_variance <- MCMCglmm.variance(model)
boxplot(terms_variance, horizontal = TRUE, las = 1)
```

`model.test` *Model Test*

Description

Fit models of disparity change through time

Usage

```
model.test(
  data,
  model,
  pool.variance = NULL,
  time.split = NULL,
  fixed.optima = FALSE,
  control.list = list(fnscale = -1),
  verbose = TRUE
)
```

Arguments

`data`	A `dispRity` object used to test models of evolution through time.
`model`	The model(s) of evolution to allow for changes in disparity-through-time using a homogenous or hetergenous model, either using a single input or a list containing different models (See **Details**). If a vector with multiple modes is supplied then the model will test for shifts in modes at the time supplied by `time.split`.
`pool.variance`	If `NULL` (default) the difference in variances will be calculated using `bartlett.test` of equal variances. If there is no significant difference among variances, then variance in samples will be pooled and the same variance will be used for all samples. A significance difference will not pool variances and the original variance will be used for model-testing. If argument `TRUE` or `FALSE` are used, Bartlett's test will be ignored and the analyses will use the user-set pooling of variances.

time.split	The age of the change in mode (numeric). The age is measured in positive units as the time before the most recent sample, and multiple ages can be supplied in a vector. If no age is supplied for models then all possible time shifts are fit in the model, and the highest likelihood model is returned. Note this only applies to heterogenous models (See **Details**).
fixed.optima	A logical value, whether to use an estimated optimum value in OU models (FALSE - default), or whether to set the OU optimum to the ancestral value (TRUE).
control.list	A list of fine-tune control inputs for the optim function.
verbose	logical, whether to display the model results while they are computed (TRUE - default).

Details

DISCLAIMER: this function is working properly (i.e. it does what it is supposed to do), however, the interpretation of the results has not yet been thought through, discussed and peer-reviewed (what does a Brownian motion like disparity curve means biologically?).

The models are fit using maximum likelihood optimisation using the function optim. Fine-tuning of the search algorithms can be applied using the control.list argument. Models can be fit using a homogenous model with the same process applied to the entire sequence or models with time splits that represent a change in parameters or a shift in mode. When a heterogeneous and/or a time-shift model is specified with a specified time.split then the shift is tested at that value only. If no time shift is supplied then multiple shift times are tested, with all bins that allow for at least 10 bins either side of the split. If the entire sample is fewer than 30 samples long then no time splits are searched for (unless a time split is supplied by the user). Parameters are shared across different modes. For example, c("BM", "OU") would fit a model in which the process starts with a BM model and shifts to an OU process. The ancestral value at the start of the sequence and sigma squared value are shared across the models. Any combination of the following homogenous models (with the exception of "multi.OU") can be fit to the data:

- BM: Fits a unbiased random walk model of Brownian motion evolution (Felsenstein 1973; 1985; Hunt 2006). The model optimises the ancestral state and the 'step-variance' (sigma-squared).
- OU: The Ornstein-Uhlenbeck model of evolution in which the change in variance is constrained to an optimum value (Hansen 1997). In this model there are three parameters: optima, alpha, and ancestral state. The strength of attraction based on the parameter alpha and the ancestral state is estimated from the data. The optima value is estimated from the data, and this can lead to optima being found outside the known data values, and thus the model can resemble a trend. If the argument fixed.optima = TRUE, the model will not estimate optima but constrain it to the first (ancestral) value in the sequence as is done in phylogenetic OU models.
- Trend: Fits a Brownian motion model with a directional component. This model is also known as the General Random Walk (Hunt 2006). This model has three parameters: the ancestral state, the 'step-variance' (sigma-squared), and the positive or negative trend.
- Stasis: Fits a model in which traits evolve with variance (omega) around a mean (theta). This model is time-independent in that the model is guided only by the variance and attraction to the mean (Hunt 2006).

- EB: Early-Burst, trait variance accumulates early in the evolution of a trait and decreases exponentially through time (Blomberg et al. 2003; Harmon et al. 2010). This model has three parameters: ancestral state, sigma-squared, and the exponential rate of decrease. Note this model expects the mean to remain unchanged through the model, so does not explicitly model a rapid change to a new mean or optimum value.
- multi.OU: Fits a model in which the value of the optima shifts at one or more time splits. The values of the 'step-variance' (sigma squared) and attraction to the optima (alpha) are shared across all the samples. This model can not be fit with other models - the multi.OU system can be fit to the model only.

Value

A list of class `dispRity` and `model.test` that can be plotted and summarised via `summary.dispRity` and `plot.dispRity`. The list is composed of:

- `$aic.models` summary for each model's small sample Akaike Information Criterion (AICc), delta AICc, and AICc weight
- `$full.models` the list of the full models outputs from `optim` with the estimated parameters, log-likelihood, convergence statistics, and the split.time if applicable
- `$call` the model input
- `$models.data` input data used by the model(s)
- `$fixed.optima` Logical indicating whether a fixed optima was assumed for OU model(s)

Author(s)

Mark N Puttick and Thomas Guillerme

References

Blomberg SP, Garland T Jr, & Ives AR. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. Evolution. **57**, 717-745.

Hansen TF. 1997. Stabilizing selection and the comparative analysis of adaptation. Evolution. **51**, 1341-1351.

Harmon LJ, *et al*. 2010. Early bursts of body size and shape evolution are rare in comparative data. **64**, 2385-2396.

Hunt G. 2006. Fitting and comparing models of phyletic evolution: random walks and beyond. Paleobiology. **32**, 578-601. DOI: 10.1666/05070.1.

Hunt G, Hopkins MJ & Lidgard S. 2015. Simple versus complex models of trait evolution and stasis as a response to environmental change. Proceedings of the National Academy of Sciences. **112**, 4885-4890. DOI: 10.1073/pnas.1403662111

Felsenstein J. 1973. Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics. **25**, 471-492.

Felsenstein J. 1985. Phylogenies and the comparative method. The American Naturalist. **51**, 1-15.

Murrell DJ. 2018. A global envelope test to detect non-random bursts of trait evolution. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.13006

See Also

model.test.wrapper, model.test.sim, summary.dispRity and plot.dispRity

Examples

```
## Not run:
## Mammal disparity through time
data(BeckLee_disparity)

## The four models to fit
models <- list("BM", "OU", "multi.OU", c("BM", "OU"))

## Fitting the four models to the disparity data
tests <- model.test(BeckLee_disparity, models, time.split = 66)

## Summarising the models
summary(tests)

## Plotting only the models support
plot(tests)

## End(Not run)
```

---

model.test.sim *Simulate Model Test*

Description

Simulate models of disparity change through time

Usage

```
model.test.sim(
  sim = 1,
  model,
  model.rank = 1,
  alternative = "two-sided",
  time.split = NULL,
  time.span = 100,
  variance = 1,
  sample.size = 100,
  parameters = list(),
  fixed.optima = FALSE
)
```

Arguments

sim	The number of separate simulations to run.
model	Either (i) the named model of evolution to simulate for changes in disparity-through-time using a homogenous or hetergenous model (see list in model.test) or (ii) an object of class dispRity returned from model.test function. If a dispRity object is supplied, all remaining arguments apart from sim and model.rank and alternative are ignored as the model specified by the input model is used.
model.rank	If a dispRity object is supplied, which model is used for simulation. The rank refers to the order of models as specified by AICc, so if model.rank = 1 (default) the best-fitting model is used for simulation.
alternative	If the simulation is based on a dispRity object, what is the alternative hypothesis: can be "two-sided" (default), "greater" or "lesser".
time.split	The age of the change in mode. The age is measured as the time before the most recent sample, and multiple ages can be supplied in a vector. Note this only applies to heterogenous models.
time.span	The length of the sequence (numeric). If one number is supplied this is treated as the length of the sequence and the time span is treated as sequence from 0 to time.span in unit increments. If a vector of length > 1 is supplied, this is treated as the the age of each sample in the sequence.
variance	The variance of each sample (numeric). If one number is supplied this is the variance for all samples in the sequence. If a vector of equal length to the time.span vector is supplied, this is used for the variance of each sample in the sequence
sample.size	The sample size of each sample (numeric). If one number is supplied this is the sample size for all samples in the sequence. If a vector of equal length to the time.span vector is supplied, this is used for the sample size of each sample in the sequence
parameters	A list of model parameters used for simulations. See details.
fixed.optima	A logical value, whether to use an estimated optimum value in OU models (FALSE - default), or whether to set the OU optimum to the ancestral value (TRUE).

Details

DISCLAIMER: this function is working properly (i.e. it does what it is supposed to do), however, the interpretation of the results has not yet been thought through, discussed and peer-reviewed (what does a Brownian motion like disparity curve means biologically?).

The parameters is a list of arguments to be passed to the models. These arguments can be:

- ancestral.state, ancestral value of the disparity applicable to all models (default = 0.01).
- sigma.squared, rate of step variance to all models except Stasis (default = 1).
- alpha, strength of attraction to the optimum in OU models (default = 1).
- optima.1, the value of the optimum in a OU model, or the first bin optimum in a multi-OU model (default = 0.15).

- optima.2, the second bin optimum in a multi-OU model (default = 0.15).
- optima.3, the third bin optimum in a multi-OU model (default = 0.15).
- theta.1, the mean in a Stasis model, or the first bin mean in a multi-Stasis model (default = 1).
- theta.2, the second bin optimum in a multi-OU model (default = 1).
- theta.3, the third bin optimum in a multi-OU model (default = 1).
- omega, the variance in a Stasis model (default = 1).
- trend, the trend parameter in the Trend model (default = 0.5).
- eb.rate, the rate of exponential rate decrease in the EB model (default = -0.1).

Value

A list of class dispRity and model.sim. Each list element contains the simulated central tendency, as well as the variance, sample size, and subsets used to simulate the data.

Author(s)

Mark N Puttick and Thomas Guillerme

References

Blomberg SP, Garland T Jr, & Ives AR. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. Evolution. **57**, 717-745.

Hansen TF. 1997. Stabilizing selection and the comparative analysis of adaptation. Evolution. **51**, 1341-1351.

Harmon LJ, *et al*. 2010. Early bursts of body size and shape evolution are rare in comparative data. **64**, 2385-2396.

Hunt G. 2006. Fitting and comparing models of phyletic evolution: random walks and beyond. Paleobiology. **32**, 578-601. DOI: 10.1666/05070.1.

Hunt G, Hopkins MJ & Lidgard S. 2015. Simple versus complex models of trait evolution and stasis as a response to environmental change. Proceedings of the National Academy of Sciences. **112**, 4885-4890. DOI: 10.1073/pnas.1403662111

Felsenstein J. 1973. Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics. **25**, 471-492.

Felsenstein J. 1985. Phylogenies and the comparative method. The American Naturalist. **51**, 1-15.

Murrell DJ. 2018. A global envelope test to detect non-random bursts of trait evolution. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.13006

Citation for the envelope code:

See Also

model.test, model.test.wrapper, summary.dispRity and plot.dispRity

Examples

```
## Not run:
## Disparity through time data
data(BeckLee_disparity)

## List of models to test
models <- list("Trend", "BM")

## Testing the models on the observed disparity
model_test_output <- model.test(BeckLee_disparity, models, time.split = 66)

## simulations using the output from model.test
model_test_sim_output <- model.test.sim(sim = 100, model= model_test_output)

## Plot the simulated best model
plot(model_test_sim_output)
## Add the observed data
plot(BeckLee_disparity, add = TRUE, col = c("pink", "#ff000050", "#ff000050"))

## Simulating a specific model with specific parameters parameters
model_simulation <- model.test.sim(sim = 100, model = "BM", time.span = 120, variance = 0.1,
                                   sample.size = 100, parameters = list(ancestral.state = 0,
                                       sigma.squared = 0.1))

## Summarising the results
plot(model_simulation, main = "A simple Brownian motion")

## End(Not run)
```

`model.test.wrapper` *Model test wrapper*

Description

A wrapper function for `model.test` to perform a model fitting analysis on disparity through time data.

Usage

```
model.test.wrapper(
  data,
  model,
  pool.variance = NULL,
  time.split = NULL,
  fixed.optima = FALSE,
  control.list = list(fnscale = -1),
  verbose = TRUE,
  sim = 1000,
```

```
  plot.sim = TRUE,
  col.sim,
  col.obs = "hotpink",
  lwd.obs = 2,
  show.p = FALSE,
  cex.p,
  legend = FALSE,
  ...
)
```

Arguments

data	A dispRity object used to test models of evolution through time.
model	The model(s) of evolution to allow for changes in disparity-through-time using a homogenous or hetergenous model, either using a single input or a list containing different models (see list in model.test). If a vector with multiple modes is supplied then the model will test for shifts in modes at the time supplied by time.split.
pool.variance	If NULL (default) the difference in variances will be calculated using bartlett.test of equal variances. If there is no significant difference among variances, then variance in samples will be pooled and the same variance will be used for all samples. A significance difference will not pool variances and the original variance will be used for model-testing. If argument TRUE or FALSE are used, Bartlett's test will be ignored and the analyses will use the user-set pooling of variances.
time.split	The age of the change in mode (numeric). The age is measured in positive units as the time before the most recent sample, and multiple ages can be supplied in a vector. If no age is supplied for models then all possible time shifts are fit in the model, and the highest likelihood model is returned. Note this only applies to heterogenous models (See **Details**).
fixed.optima	A logical value, whether to use an estimated optimum value in OU models (FALSE - default), or whether to set the OU optimum to the ancestral value (TRUE).
control.list	A list of fine-tune control inputs for the optim function.
verbose	logical, whether to display the model results as computed (TRUE - default).
sim	The number of separate simulations (default = 1000).
plot.sim	Logical. If TRUE (default) the plots of the simulated and observed disparity are returned for all models.
col.sim	Colour options used for the plotting of simulated values. See plot.dispRity for more details. If missing, the default colours c("black", "lightgrey", "grey") are used.
col.obs	Colour of the observed data on the plot. Default colour is "hotpink"
lwd.obs	Line width of the observed value.
show.p	Logical, when plot.sim = TRUE, whether to display the p-value of rank envelope tests (TRUE) or not (FALSE - default).

cex.p	A numerical value for the the font size of the displayed p-value (if show.p = TRUE). If missing, the value is set to 1.
legend	Logical, when plot.sim = TRUE, whether to display the legend in the first panel (TRUE) or not (FALSE - default).
...	Any additional arguments to be passed to plot.dispRity or summary.dispRity.

Details

This function gives the relative fit of model.test output using log-likelihood and AICc values, as well as the Rank Envelope Test significance to elucidate if empirical data is significantly different to simulated data modelled using the estimated model parameters from model.test.sim. This is equivalent to running test <- model.test.sim(sim = 1000, model = model.test(data, model)); summary(test) ; plot(test) ; plot(data, add = TRUE).

DISCLAIMER: this function is working properly (i.e. it does what it is supposed to do), however, the interpretation of the results has not yet been thought through, discussed and peer-reviewed (what does a Brownian motion like disparity curve means biologically?).

Value

A matrix with the relative fit, parameter values, and Rank Envelope test p values for each model, and a plot of simulated data from each model alongside observed data for each model if plot.sim is TRUE

Author(s)

Mark N Puttick and Thomas Guillerme

References

Blomberg SP, Garland T Jr, & Ives AR. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. Evolution. **57**, 717-745.

Hansen TF. 1997. Stabilizing selection and the comparative analysis of adaptation. Evolution. **51**, 1341-1351.

Harmon LJ, *et al*. 2010. Early bursts of body size and shape evolution are rare in comparative data. **64**, 2385-2396.

Hunt G. 2006. Fitting and comparing models of phyletic evolution: random walks and beyond. Paleobiology. **32**, 578-601. DOI: 10.1666/05070.1.

Hunt G, Hopkins MJ & Lidgard S. 2015. Simple versus complex models of trait evolution and stasis as a response to environmental change. Proceedings of the National Academy of Sciences. **112**, 4885-4890. DOI: 10.1073/pnas.1403662111

Felsenstein J. 1973. Maximum-likelihood estimation of evolutionary trees from continuous characters. American Journal of Human Genetics. **25**, 471-492.

Felsenstein J. 1985. Phylogenies and the comparative method. The American Naturalist. **51**, 1-15.

Murrell DJ. 2018. A global envelope test to detect non-random bursts of trait evolution. Methods in Ecology and Evolution. DOI: 10.1111/2041-210X.13006

See Also

model.test, model.test.sim, summary.dispRity and plot.dispRity

Examples

```
## Not run:
## Mammal disparity through time
data(BeckLee_disparity)

## The models to be fit to disparity data
models <- list("BM", "OU", "multi.OU", "Trend")

## test all models, and assess the significance of simulated data
## against the empirical distribution for each
model.test.wrapper(data = BeckLee_disparity, model = models, fixed.optima = TRUE,
                   time.split = 66, show.p = TRUE)

## End(Not run)
```

---

multi.ace *Ancestral states estimations with multiple trees*

Description

Fast ancestral states estimations run on multiple trees using the Mk model from castor::asr_mk_model.

Usage

```
multi.ace(
  data,
  tree,
  models,
  threshold = TRUE,
  special.tokens,
  special.behaviours,
  brlen.multiplier,
  verbose = FALSE,
  parallel = FALSE,
  output,
  options.args,
  estimation.details = NULL
)
```

Arguments

data	A matrix, data.frame or list with the characters for each taxa.
tree	A phylo or mutiPhylo object (if the tree argument contains node labels, they will be used to name the output).
models	A character vector, unambiguous named list or matrix to be passed as model arguments to castor::asr_mk_model or ape::ace (see details).
threshold	either logical for applying a relative threshold (TRUE - default) or no threshold (FALSE) or a numeric value of the threshold (e.g. 0.95). See details.
special.tokens	optional, a named vector of special tokens to be passed to grep (make sure to protect the character with "\\"). By default special.tokens <- c(missing = "\\?", inapplicable = "\\-", polymorphism = "\\&", uncertainty = "\\/"). Note that NA values are not compared and that the symbol "@" is reserved and cannot be used.
special.behaviours	optional, a list of one or more functions for a special behaviour for special.tokens. See details.
brlen.multiplier	optional, a vector of branch length modifiers (e.g. to convert time branch length in changes branch length) or a list of vectors (the same length as tree).
verbose	logical, whether to be verbose (TRUE) or not (FALSE - default).
parallel	Either a logical, whether to use parallel algorithm (TRUE) or not (FALSE - default); or directly an integer indicating the number of cores to use (note that if parallel = 1, one core will be used but the parallel integration will still be called).
output	optional, see Value section below.
options.args	optional, a named list of options to be passed to function called by castor::asr_mk_model.
estimation.details	optional, whether to also return the details for each estimation as returned by castor::asr_mk_model or ape::ace. This argument can be left NULL (default) or be any combination of the elements returned by castor::asr_mk_model or ape::ace (e.g. c("loglikelihood", "transition_matrix", "CI95")).

Details

Depending on the type of characters models argument can be either:

- the name of a single model to apply to all characters (if all characters are discrete or all are continuous); see below for the list of available names. For example models = "ER" applies the Equal Rates model to all characters (assuming they are all discrete characters).
- a vector of model names to apply to different type of characters (see below for the list). For example models = c("ER", "ER", "BM") applies the Equal Rates model to the two first characters (discrete) and the "BM" model to the third character (continuous).
- a transition "matrix" to be applied to all characters (if discrete). For example models = matrix(0.2, 2, 2).

- an single named list of arguments to be applied to all characters by passing it to `ape::ace` (if continuous). For example `models = list(method = "GLS", corStruct = corBrownian(1, my_tree))`.
- an un-ambiguous list of arguments to be passed to either `castor::asr_mk_model` (discrete characters) or `ape::ace` (continuous characters). For example `models = list("char1" = list(transition_matrix = matrix(0.2, 2, 2)), "char2" = list(method = "GLS", corStruct = corBrownian(1, my_tree)))` to be specifically passed to the characters named "char1" and "char2"

The available built-in models for discrete characters in `castor::asr_mk_model` are:

- `"ER"` for all equal rates
- `"SYM"` for symmetric rates
- `"ARD"` all rates are different
- `"SUEDE"` equal stepwise transitions (e.g. for meristic/counting characters)
- `"SRD"` different stepwise transitions

See directly `castor::asr_mk_model` for more models.

The available built-in models and methods for continuous characters in `ape::ace` are:

- `"BM"` model: for a default Brownian Motion with the "REML" method
- `"REML"` method: for a default Brownian Motion with the "REML" method (same as above)
- `"ML"` method: for a default Brownian Motion with the "ML" method
- `"pic"` method: for a default Brownian Motion with the "pic" (least squared) method

The `threshold` option allows to convert ancestral states likelihoods into discrete states. When `threshold = FALSE`, the ancestral state estimated is the one with the highest likelihood (or at random if likelihoods are equal). When `threshold = TRUE`, the ancestral state estimated are all the ones that are have a scaled likelihood greater than the maximum observed scaled likelihood minus the inverse number of possible states (i.e. `select_state >= (max(likelihood) - 1/n_states)`). This option makes the threshold selection depend on the number of states (i.e. if there are more possible states, a lower scaled likelihood for the best state is expected). Finally using a numerical value for the threshold option (e.g. `threshold = 0.95`) will simply select only the ancestral states estimates with a scaled likelihood equal or greater than the designated value. This option makes the threshold selection absolute. Regardless, if more than one value is select, the uncertainty token (`special.tokens["uncertainty"]`) will be used to separate the states. If no value is selected, the uncertainty token will be use between all observed characters (`special.tokens["uncertainty"]`).

`special.behaviours` allows to generate a special rule for the `special.tokens`. The functions should can take the arguments `character, all_states` with `character` being the character that contains the special token and `all_states` for the character (which is automatically detected by the function). By default, missing data returns and inapplicable returns all states, and polymorphisms and uncertainties return all present states.

- `missing = function(x,y) y`
- `inapplicable = function(x,y) y`
- `polymorphism = function(x,y) strsplit(x, split = "\\&")[[1]]`
- `uncertainty = function(x,y) strsplit(x, split = "\\/")[[1]]`

Functions in the list must be named following the special token of concern (e.g. `missing`), have only `x, y` as inputs and a single output a single value (that gets coerced to `integer` automatically). For example, the special behaviour for the special token `"?"` can be coded as: `special.behaviours = list(missing = function(x, y) return(NA)` to make ignore the character for taxa containing `"?"`.

When using the parallel option (either through using `parallel = TRUE` by using the number of available cores minus on or manually setting the number of cores - e.g. `parallel = 5`), the `castor::asr_mk_model` function will use the designated number of cores (using the option `Nthreads = <requested_number_of_cores>`). Additionally, if the input `tree` is a `"multiPhylo"` object, the trees will be run in parallel for each number of cores, thus decreasing computation time accordingly (e.g. if 3 cores are requested and `tree` contains 12 `"phylo"` objects, 4 different `"phylo"` objects will be run in parallel on the 3 cores making the calculation around 3 times faster).

Value

Returns a `"matrix"` or `"list"` of ancestral states. By default, the function returns the ancestral states in the same format as the input `matrix`. This can be changed using the option `output = "matrix"` or `"list"` to force the class of the output. To output the combined ancestral states and input, you can use `"combined"` (using the input format) or `"combined.matrix"` or `"combined.list"`. If using continuous characters only, you can use the output option `"dispRity"` to directly output a usable `dispRity` object with all trees and all the data (estimated and input). *NOTE* that if the input data had multiple character types (continuous and discrete) and that `"matrix"` or `"combined.matrix"` output is requested, the function returns a `"data.frame"`.

Author(s)

Thomas Guillerme

See Also

`castor::asr_mk_model`, `char.diff`

Examples

```
set.seed(42)
## A simple example:
## A random tree with 10 tips
tree <- rcoal(10)
## Setting up the parameters
my_rates = c(rgamma, rate = 10, shape = 5)

## A random Mk matrix (10*50)
matrix_simple <- sim.morpho(tree, characters = 50, model = "ER", rates = my_rates,
                            invariant = FALSE)

## Run a basic ancestral states estimations
ancestral_states <- multi.ace(matrix_simple, tree)
ancestral_states[1:5, 1:5]

## A more complex example
```

```
## Create a multiple list of 5 trees
multiple_trees <- rmtree(5, 10)

## Modify the matrix to contain missing and special data
matrix_complex <- matrix_simple
matrix_complex[sample(1:length(matrix_complex), 50)] <- "-"
matrix_complex[sample(1:length(matrix_complex), 50)] <- "0%2"
matrix_complex[sample(1:length(matrix_complex), 50)] <- "?"
matrix_complex[1:5,1:5]

## Set a list of extra special tokens
my_spec_tokens <- c("weirdtoken" = "%")

## Set some special behaviours for the "weirdtoken" and for "-" and "?"
my_spec_behaviours <- list()
## Inapplicable tokens "-" are ignored
my_spec_behaviours$inapplicable <- function(x,y) return(NA)
## Missing tokens "?" are considered as all states
my_spec_behaviours$missing      <- function(x,y) return(y)
## Weird tokens are considered as state 0 and 3
my_spec_behaviours$weirdtoken   <- function(x,y) return(c(1,2))

## Create a random branch length modifier to apply to each tree
branch_lengths <- rnorm(18)^2

## Setting a list of model ("ER" for the 25 first characters and then "SYM")
my_models <- c(rep("ER", 25), rep("SYM", 25))

## Run the ancestral states on all the tree with multiple options
ancestral_states <- multi.ace(matrix_complex, multiple_trees,
                              verbose = TRUE,
                              models = my_models,
                              threshold = 0.95,
                              special.tokens = my_spec_tokens,
                              special.behaviours = my_spec_behaviours,
                              brlen.multiplier = branch_lengths,
                              output = "combined.matrix")

## The results for the the two first characters for the first tree
ancestral_states[[1]][, 1:2]

## Not run:
## The same example but running in parallel
ancestral_states <- multi.ace(matrix_complex, multiple_trees,
                              verbose = TRUE,
                              models = my_models,
                              threshold = 0.95,
                              special.tokens = my_spec_tokens,
                              special.behaviours = my_spec_behaviours,
                              brlen.multiplier = branch_lengths,
                              output = "combined.matrix",
                              parallel = TRUE)
```

```
## End(Not run)
```

`null.test` *Testing a null hypothesis on multidimensional data.*

Description

Testing the difference between the observed disparity and disparity under a null model.

Usage

```
null.test(
  data,
  replicates = 100,
  null.distrib,
  null.args = NULL,
  null.cor = NULL,
  null.scree = NULL,
  alter = "two-sided",
  scale = FALSE,
  ...
)
```

Arguments

`data`	a `dispRity` object.
`replicates`	the number of replicates for the test (default = `100`).
`null.distrib`	one or more distribution functions to generate the null model to be passed to `space.maker`.
`null.args`	any additional distribution arguments to be passed to `space.maker` (see `arguments` within; `default = NULL`).
`null.cor`	an additional correlation matrix to be passed to `space.maker` (see `cor.matrix` within; `default = NULL`).
`null.scree`	an additional vector of variance per axis (equivalent to screeplot output); `default = NULL`).
`alter`	the type of alternative hypothesis (H1) as used in `randtest` (`default = "two-sided"`).
`scale`	whether to scale the simulated and the observed data.
`...`	optional arguments to be passed to `as.randtest`.

Author(s)

Thomas Guillerme

References

Diaz, S., Kattge, J., Cornelissen, J.H., Wright, I.J., Lavorel, S., Dray, S., Reu, B., Kleyer, M., Wirth, C., Prentice, I.C. and Garnier, E., **2016**. The global spectrum of plant form and function. Nature, 529(7585), pp.167-171.

See Also

space.maker, test.dispRity

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_mat50)
## Calculating the disparity as the ellipsoid volume
obs_disparity <- dispRity(BeckLee_mat50, metric = ellipsoid.volume)
## Testing against normal distribution
results <- null.test(obs_disparity, replicates = 100, null.distrib = rnorm)
results ; plot(results)

## Running the test on multiple subsets (may take some time!)
## Generating the subsets
groups <- as.data.frame(matrix(data = c(rep(1, 12), rep(2, 13), rep(3, 12),
     rep(4, 13)), dimnames = list(rownames(BeckLee_mat50))), ncol = 1)
customised_subsets <- custom.subsets(BeckLee_mat50, groups)
## Bootstrapping the data
bootstrapped_data <- boot.matrix(customised_subsets, bootstraps = 100)
## Calculating variances of each dimension
sum_variances <- dispRity(bootstrapped_data, metric = c(sum, variances))
## Testing against normal distribution
results <- null.test(sum_variances, replicates = 100, null.distrib = rnorm)
summary(results) ; plot(results)
```

---

pair.plot *Plots pairwise comparisons*

Description

Plots pairwise comparisons from a data frame (typically output from test.dispRity).

Usage

```
pair.plot(
  data,
  what,
  col = c("black", "white"),
  legend = FALSE,
  binary,
  diag,
```

```
  add,
  lower = TRUE,
  ...
)
```

Arguments

data	A matrix or a data.frame object with comparisons' pair names as row names. The number of rows must be equal to a pairwise combination of n elements (see details).
what	A numeric or character value designating which column to plot.
col	The two extremes of a color gradient (default = c("black", "white")).
legend	Logical, whether to plot the legend or not.
binary	Optional, if the results must be binary, a numeric value for the threshold of acceptance (values greater will be 1, lower will be 0).
diag	Optional, can be "max" or "min" or a single numeric value.
add	Optional, whether to add significance tokens can be numeric for a point type to print (pch) or "character" to print (e.g. "*").
lower	Optional, logical, whether to add tokens for values lower than binary (default is TRUE; FALSE will add tokens for values bigger than binary).
...	Any other options to be passed to plot.

Details

The number of rows (i.e. comparisons) in matrix must be equal to the results of a pairwise combination. In general, the number of rows x must satisfy the equation: $x = n^2/2 - n/2$ where n must be an integer greater or equal than 2.

Author(s)

Thomas Guillerme

See Also

test.dispRity.

Examples

```
## A small matrix of two pairwise comparisons of seven elements (2*21 comparisons)
data <- matrix(data = runif(42), ncol = 2)

## Plotting the first column as a pairwise comparisons
pair.plot(data, what = 1, col = c("orange", "blue"), legend = TRUE, diag = 1)

## Adding some tokens for each value below 0.2 in the second column
pair.plot(data, what = 2, binary = 0.2, add = "*", cex = 2)

## Loading disparity data
```

```
data(disparity)

## Testing the pairwise difference between slices
tests <- test.dispRity(disparity, test = wilcox.test, correction = "bonferroni")

## Plotting the significance
pair.plot(as.data.frame(tests), what = "p.value", binary = 0.05)
```

`pgls.dispRity` *phylolm dispRity (from* `phylolm::phylolm`*)*

Description

Passing `dispRity` objects to the `phylolm` function from the `phylolm` package. Typically to run some PGLS.

Usage

```
pgls.dispRity(data, tree, formula, model = "BM", ..., optim = list())
```

Arguments

`data`	A `dispRity` object with a metric of dimension level 2 at least
`tree`	If `data` does not contain a tree component, a `"phylo"` or `"multiPhylo"` object to be used as the tree. If `data` already contains a tree component and the `tree` argument is not missing, the provided `tree` will replace any contained in `data`.
`formula`	The PGLS formula. If left empty, runs either `disparity ~ 1` or `disparity ~ subsets` if `data` contains subsets.
`model`	The covariance model (default is `"BM"`). For more details (including the models available) see the manual for `phylolm`.
`...`	Any optional arguments to be passed to `phylolm`
`optim`	An optional named list of arguments to be passed to the function `optim`

Details

The `formula` needs to be expressed by always naming the response variable `disparity` to use the calculated disparity data from `data`.

Optional arguments `...` correspond to all the non-ambiguous named arguments from the `phylolm`. Optional arguments for the internal `optim` function can be passed as a named list to the `optim` argument.

Author(s)

Thomas Guillerme

See Also

phylolm, test.dispRity, custom.subsets, chrono.subsets.

Examples

```
## Simple example
data(BeckLee_mat50)
data(BeckLee_tree)
disparity <- dispRity(BeckLee_mat50, metric = centroids, tree = BeckLee_tree)

## Running a simple PGLS
model <- pgls.dispRity(disparity)
summary(model)

## More complex example running a PGLS
## on multiple trees and using groups as a predictor
```

---

plot.char.diff *Plots character differences*

Description

Plots a character difference matrix from a discrete character matrix or its character differences density profile.

Usage

```
## S3 method for class 'char.diff'
plot(
  x,
  ...,
  type = "matrix",
  legend = TRUE,
  legend.title = "Difference",
  legend.pos = "topleft",
  legend.round = 0,
  axis = TRUE,
  xlim,
  ylim,
  xlab,
  ylab,
  col,
  main
)
```

Arguments

x	A discrete matrix or an already computed character difference matrix of class char.diff.
...	Any additional graphical arguments to be passed to image.
type	Either "matrix" (or "m") or "density" (or "d") for respectively plotting the matrix of character differences or its character differences density profile.
legend	A logical value stating whether to print the legend or not (default = TRUE).
legend.title	A character string to be displayed as the title of the legend (default = Difference).
legend.pos	The position of the legend. Can be two numeric. Default is "topleft".
legend.round	A numeric value for digits up legend values. Default is 0.
axis	A logical value stating whether to print the axis or not (default = TRUE).
xlim	Two numeric values to determine the x axis limits. If missing (default), the limits are calculated automatically to fit the plot window.
ylim	Two numeric values to determine the y axis limits. If missing (default), the limits are calculated automatically to fit the plot window.
xlab	A character string for the the x axis. Can be missing.
ylab	A character string for the the y axis. Can be missing.
col	Two colors for forming the gradient if type = "correlation" or for the density lines colors if type = "density".
main	An overall title for the plot.

Author(s)

Thomas Guillerme

See Also

char.diff

Examples

```
## Comparing two characters
char.diff(list(c(0, 1, 0, 1), c(0, 1, 1, 1)))

## Pairwise comparisons in a morphological matrix
morpho_matrix <- matrix(sample(c(0,1), 100, replace = TRUE), 10)

## Plotting a matrix
plot.char.diff(morpho_matrix)

## Plotting the density profile of a char.diff object
char.diff_matrix <- char.diff(morpho_matrix)
plot(char.diff_matrix, type = "density")
```

plot.dispRity *dispRity object plotting*

Description

Plots a dispRity object.

Usage

```
## S3 method for class 'dispRity'
plot(
  x,
  ...,
  type,
  quantiles = c(50, 95),
  cent.tend = median,
  rarefaction = NULL,
  elements = FALSE,
  observed = FALSE,
  add = FALSE,
  density = NULL,
  specific.args
)
```

Arguments

x	A dispRity object.
...	Any optional arguments to be passed to plot. See details.
type	Either "continuous", "box", "line", "polygon" or "space". When unspecified, if no disparity was calculated, "preview" is used. If disparity was calculated, "continuous" is used for chrono.subsets and "box" for custom.subsets. See details.
quantiles	The quantiles to display (default is quantiles = c(50, 95); is ignored if the dispRity object is not bootstrapped).
cent.tend	A function for summarising the bootstrapped disparity values (default is median).
rarefaction	Either NULL (default) or FALSE for not using the rarefaction scores; a numeric value of the level of rarefaction to plot; or TRUE for plotting the rarefaction curves.
elements	logical whether to plot the number of elements per subsets (default is FALSE) or a list of any of the graphical arguments "col", "pch" and/or "cex".
observed	logical whether to add the observed values on the plot as crosses (default is FALSE) or a list of any of the graphical arguments "col", "pch" and/or "cex".
add	logical whether to add the new plot an existing one (default is FALSE).
density	the density of shading lines to be passed to polygon. Is ignored if type = "box" or type = "line".

`specific.args` optional, a named list of arguments to be passed for some specific plot types. See details.

Details

When specifying optional arguments with `...` in a graph with multiple elements (e.g. `points`, `lines`, etc...) you can specify which specific element to affect using the syntax `<element>.<argument>`. For example if you want everything in the plot to be in blue at the exception of the points to be red, you can use `plot(..., col = "blue", points.col = "red")`.

The different `type` arguments are:

- `"continuous"`: plots the results as a continuous line.
- `"box"`: plots the results as discrete box plots (note that this option ignores the user set quantiles and central tendency).
- `"line"`: plots the results as discrete vertical lines with the user's set quantiles and central tendency.
- `"polygon"`: identical as `"line"` but using polygons rather than vertical lines.
- `"preview"`: plots two dimensional preview of the space (default is `c(1,2)`). WARNING: the plotted dimensions might not be representative of the full multi-dimensional space!

The different `specific.args` arguments for the following options are:

- if `type = "preview"`, the specific arguments can be:
  - `dimensions`: two specific dimensions to plot (default is `specific.args = list(dimensions = c(1,2))`;
  - `matrix`: which specific matrix to plot the data from (by default, all the matrices are used).
  - `tree`: whether to plot the underlying tree(s) or not. Can be either logical, whether to plot no tree (default is `specific.args = list(tree = FALSE)`), all trees (`specific.args = list(tree = TRUE)`) or a specific set of trees (e.g. `specific.args = list(tree = c(1,2))`)
- if data is a `"test.metric"` result that was obtained with the option `save.steps = TRUE` (see `test.metric`), it is possible to specify which steps to by specifying the following specific argument: `specific.args = list(visualise.steps = c(1,4,5))` for visualising steps 1, 4 and 5 of the different shifts. By default, if the `"test.metric"` was obtained with the option `save.steps = TRUE`, four steps are displayed.
- if data is a `"dispRity"` and `"projection"` object (from `dispRity.covar.projections`), it is possible to plot either the boxplot of disparity values for each projection (using `correlation.plot = NULL`; default) or to plot the correlation between two calculated elements (e.g. `correlation.plot = c("position", "distance")`).

When plotting `"randtest"` or `"test.metric"` data or when using `type = "preview"` a legend is plotted by default. To remove the legend you can use the argument `legend = FALSE`. You can control specific arguments for the legend using the `...` optional arguments preceded by `legend.`. For example, to change the legend position you can use `legend.x = "topleft"` or `legend.x = 4.2` and `legend.y = 1.23`. General `legend` arguments such as `col`, `legend`, `pch`, etc... are recycled by the function but can always be specified using this syntax.

Author(s)

Thomas Guillerme

See Also

dispRity, summary.dispRity, null.test, dtt.dispRity, model.test, model.test.sim, test.metric

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## Discrete plotting
plot(disparity, type = "box")

## Using polygons rather than boxes (quantiles and central tendency can be
## set by the user)
plot(disparity, type = "polygon", quantiles = c(10, 50, 95),
     cent.tend = mean)

## Using different options
plot(disparity, type = "line", elements = TRUE, ylim = c(0, 3),
     xlab = ("Time (Ma)"), ylab = "disparity")

## Continuous plotting (all default options)
plot(disparity, type = "continuous")

## Rarefactions plots
plot(disparity, rarefaction = TRUE)

## Observed data
plot(disparity, observed = TRUE)

## Observed data with graphical details
plot(disparity, observed = list("pch" = 19, col = "blue", cex = 4))

## For plotting dispRity objects with the dual classes "randtest", "dtt",
## "model.test", "model.sim" and "test.metric" see the examples
## in the specific function manuals from the "See also" section above
```

---

print.dispRity *Prints a* dispRity *object.*

Description

Summarises the content of a dispRity object.

Usage

```
## S3 method for class 'dispRity'
print(x, all = FALSE, ...)
```

Arguments

x	A dispRity object.
all	logical; whether to display the entire object (TRUE) or just summarise its contents (FALSE - default).
...	further arguments to be passed to print or to print.dispRity.

Author(s)

Thomas Guillerme

See Also

custom.subsets, chrono.subsets, boot.matrix, dispRity.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## Displaying the summary of the object content
disparity
print(disparity) # the same
print.dispRity(disparity) # the same

## Displaying the full object
print.dispRity(disparity, all = TRUE)
```

---

random.circle *Random circle*

Description

Creates coordinates for a random circle

Usage

```
random.circle(n, distribution, inner = 0, outer = Inf, ...)
```

Arguments

n	The number of pairs x,y of coordinates.
distribution	The distribution from which the coordinates are sampled.
inner	Optional, the radius for an empty inner circle.
outer	Optional, the maximum radius for the circle.
...	Any additional argument to be passed to distribution.

Author(s)

Thomas Guillerme

See Also

space.maker

Examples

```
## A simple uniform circle
plot(random.circle(1000, runif), pch = 20)

## A normal ring with inner and outer boundaries
plot(random.circle(1000, rnorm, inner = 0.5, outer = 5), pch = 20)
```

---

randtest.dispRity *Random (permutation) test*

Description

Performs a random test (aka permutation test) on a matrix or a dispRity object.

Usage

```
## S3 method for class 'dispRity'
randtest(
  xtest,
  subsets,
  metric,
  replicates = 100,
  resample = TRUE,
  alter = "two-sided",
  ...
)
```

Arguments

xtest	The matrix or a dispRity object to draw from.
subsets	A vector of elements to test (or a list of vectors - see details).
metric	A function to be the statistic to apply to the subset.
replicates	A numeric value for the number of replicates (default = 100).
resample	logical whether to resample the full distribution (TRUE; default) or the distribution without the subset (FALSE).
alter	The alternative hypothesis. Can be "two-sided" (default), "greater" or "lesser".
...	optional arguments to be passed to metric.

Details

This test checks whether the metric calculated on a given subset of the data is significantly different from the metric calculated on any random subset of the same size. In other words: does the given subset have a clearly different disparity value than the rest of the data?

First, the metric (statistic) is applied to the subset sampled from the data (population). Second, the metric is applied to random equally sized subsets from the data. If the observed difference falls out of the random differences distribution, the differences are significant. This algorithm is based on a similar procedure than in link[ade4]{rantest}.

If data is a dispRity object, the subsets, metric and replicates can be left missing and are automatically inherited from the dispRity if it contains respectively subsets (from chrono.subsets or custom.subsets) a metric (from dispRity) and bootstrap draws (from boot.matrix).

If data is a dispRity object subsets can be a list of subsets to compare for example list(c("A", "B"), c("B", "A")) will run two tests comparing respectively sample A to B and B to A. *Note* that it will only compare these two samples and use their combined size as the population size, if you want to compare a subset to all the subsets you can use list(c("A") or write down the specific subsets to be used.

Value

This function returns a "randtest" object that can be passed to the generic S3 functions print.randtest or plot.randtest. The output also contains to extra elements output$observed and output$random containing the raw results of respectively the observed and random tests.

Author(s)

Thomas Guillerme

See Also

randtest

Examples

```
## Simple example
dummy_matrix <- matrix(rnorm(500), 100, 5)

## Testing whether the mean of a random subset
## is different than the means of 100 subsets
dummy_test <- randtest.dispRity(dummy_matrix,
                                subset = sample(1:100, 20),
                                metric = mean)
dummy_test ; plot(dummy_test)

## Applying this on dispRity objects
data(disparity)
test_disparity <- test.dispRity(disparity,
                                test = randtest.dispRity)

## The summarised results
summary(test_disparity)

## Plotting the results
plot(test_disparity)

## Applying this on a dispRity object with specific subset comparisons
test_disparity2 <- randtest.dispRity(disparity, subsets = list(
     ## Comparing subset 90 to the whole population (traitspace)
     c(observed = "90"),
     ## Comparing subset "70" to "90", "70" and "30"
     c(observed = "70", random = c("90", "70", "30"))))

## Summarising and plotting the results
summary(test_disparity2)
plot(test_disparity2)
```

---

reduce.matrix *Reduce a matrix*

Description

Reduce the number of rows/columns in a matrix to optimise overlap

Usage

```
reduce.matrix(matrix, distance = "gower", by.row = TRUE, verbose = FALSE)
```

Arguments

matrix	A matrix
distance	which distance to consider (passed to vegdist, default = "gower")

by.row	Whether to do it by rows (TRUE - default), or by columns (FALSE)
verbose	Whether to do be verbose (TRUE) or not (FALSE - default)

Author(s)

Thomas Guillerme

Examples

```
set.seed(1)
## A 10*5 matrix
na_matrix <- matrix(rnorm(50), 10, 5)
## Making sure some rows don't overlap
na_matrix[1, 1:2] <- NA
na_matrix[2, 3:5] <- NA
## Adding 50% NAs
na_matrix[sample(1:50, 25)] <- NA
## Illustrating the gappy matrix
image(t(na_matrix), col = "black")

## Reducing the matrix by row
(reduction <- reduce.matrix(na_matrix))
## Illustrating the overlapping matrix
image(t(na_matrix[-as.numeric(reduction$rows.to.remove), ]), col = "black")

## Reducing the matrix by columns (and being verbose)
reduce.matrix(na_matrix, by.row = FALSE, verbose = TRUE)
```

---

reduce.space *Reduce space*

Description

Remove elements from a multidimensional space

Usage

```
reduce.space(
  space,
  type,
  remove,
  parameters,
  tuning,
  verbose = FALSE,
  return.optim = FALSE
)
```

Arguments

space	the trait space
type	how to reduce the space (either "random", "size", "density", "evenness" or "position")
remove	the proportion of elements to be removed (in probability)
parameters	the parameter(s) for removal selection (see details). If left empty, the parameters is estimated to reach the amount set by remove.
tuning	Optinal parameters for tuning the parameter estimations (if remove is required and parameters is missing) a list of three parameters: "max" for the maximum of operations, "tol" for the tuning (e.g. 0.1 close), "inc.steps" for the initial increment value during optimisation (default = 2 - the bigger the value, the slower the increment).
verbose	wether to be verbose or not
return.optim	logical, whether to also return the optimal value.

Details

The type of reductions algorithms select the proportion of elements to remove (from the remove parameter). The different algorithms are:

- "random" for randomly selecting a proportion of data points (using sample(..., replace = FALSE)).
- "size" for selecting the proportion of data points closer to the centre.
- "density" for selecting the proportion of data points with the lower nearest neigbhour distances.
- "evenness" for randomly selecting the proportion of data points from the regions with most density.

The parameters for each reduction type algorithms are:

- "size" parameters: a list of parameters$centre, the centre from which to count the radius (if missing, is set to 0); and parameters$radius, the radius for removal.
- "density" parameters: a list of parameters$what "close" (default) for close neighbours or "distant" for distant ones; parameters$diameter the diameter for considering closeness or distance; parameters$output either "singles" or "pairs" to return the pairs of neighbours or one of them only (the first).
- "position" parameters: a list of parameters$value, value the threshold value from which to remove elements.
- "evenness" parameters: a list of parameters$bw, a bandwith selector function (bw.nrd0 by default); and parameters$power a scaling factor for exaggerating the flatting/narrowing of the curve (the counts are set to this parameter exponent: default is 1).

See Guillerme et al. 2020 and https://github.com/TGuillerme/moms for details.

Value

A vector of `logical` values of the rows to remove selected by the function. `TRUE` corresponds to the following (and `FALSE` to the opposite):

- "random": the randomly selected points.
- "size": the points closer to the centre of the space.
- "density": the points closer to each other.
- "position": the points on the "positive" side of the space (typically upper right corner in 2D).
- "evenness": the randomly select points from the higher density regions.

Author(s)

Thomas Guillerme

References

Guillerme T, Puttick MN, Marcy AE, Weisbecker V. **2020** Shifting spaces: Which disparity or dissimilarity measurement best summarize occupancy in multidimensional spaces?. Ecol Evol. 2020;00:1-16. (doi:10.1002/ece3.6452)

See Also

`test.metric` `dispRity`

Examples

```
set.seed(1)
## Creating a two dimensional space
space <- space.maker(100, 2, distribution = stats::rnorm)

## Generating the four types of reductions
random <- reduce.space(space, "random", remove = 0.5)
size <- reduce.space(space, "size", remove = 0.5)
density <- reduce.space(space, "density", remove = 0.5)
position <- reduce.space(space, "position", remove = 0.5)
evenness <- reduce.space(space, "evenness", remove = 0.5)

## Plotting the four different results
par(mfrow = c(3,2))
plot(space, pch = 19, col = c("grey", "black")[as.factor(random)],
     main = "Random removal")
plot(space, pch = 19, col = c("grey", "black")[as.factor(size)],
     main = "Size removal")
plot(space, pch = 19, col = c("grey", "black")[as.factor(density)],
     main = "Density removal")
plot(space, pch = 19, col = c("grey", "black")[as.factor(position)],
     main = "Position removal")
plot(space, pch = 19, col = c("grey", "black")[as.factor(evenness)],
     main = "Evenness removal")
```

```
## The space reduction with specific parameters:
# Using the point with coordinates (2,2) as the centre
# Running over a maximum of 300 iterations
# With a tolerance of 0.05 (5%)
reduce.space(space, "size", remove = 0.2,
             parameters = list("centre" = c(2,2)),
             tuning = list("max" = 300, "tol" = 0.05))

## Remove a specific amount to match a specific parameter
reduce.space(space, type = "size", parameters = list("radius" = 1.206866))
```

`remove.zero.brlen` *Remove zero branch length*

Description

Remove zero or negative branch lengths on trees by sliding nodes randomly in a postorder traversal based on `slide.nodes`.

Usage

```
remove.zero.brlen(tree, slide, verbose = FALSE)
```

Arguments

`tree`	A `"phylo"` or `"multiPhylo"` object with edge lengths
`slide`	An optional sliding `numeric` values. If left empty, 1% of the shortest branch length is used.
`verbose`	A `logical` value indicating whether to be verbose or not.

Details

The sliding value will be used to slide the nodes up and down to remove zero branch lengths by minimising the amount of branch changes. The algorithm slides the nodes up and down (when possible) on each node in a recursive way while there is still zero or negative branch lengths. If two recursions produce the same series of zero branches (e.g. by sliding node A towards node B equally so that the distance A:B becomes 0), the sliding value is divided by two until the next slide.

Value

A `"phylo"` object with a postorder edge table and no zero branch lengths.

Author(s)

Thomas Guillerme

See Also

slide.nodes

Examples

```
set.seed(42)
## Generating a tree
tree <- rtree(20)
## Adding some zero branch lengths (5)
tree$edge.length[sample(1:Nedge(tree), 5)] <- 0
any(tree$edge.length == 0) # TRUE

## And now removing these zero branch lengths!
tree_no_zero <- remove.zero.brlen(tree)
any(tree_no_zero$edge.length == 0) # FALSE

## Exaggerating the removal (to make it visible)
tree_exaggerated <- remove.zero.brlen(tree, slide = 1)

## Plot the differences
par(mfrow = c(3,1))
plot(tree, main = "zero branch length")
plot(tree_no_zero, main = "no zero branch length")
plot(tree_exaggerated, main = "exaggerated slidding")

## Removing negative branch lengths
## Generating a tree with negative branch length
set.seed(3)
tree_negative <- chronoMPL(rtree(10))
## Removing the negative branch length (and make it non-zero)
tree_positive <- remove.zero.brlen(tree_negative)
## Plot the differences
par(mfrow = c(2, 1))
plot(tree_negative, main = "Negative branch lengths")
plot(tree_positive, main = "Positive branch lengths")
```

---

scale.dispRity *Rescaling and centering disparity results.*

Description

Scales or/and centers the disparity measurements.

Usage

```
## S3 method for class 'dispRity'
scale(x, center = FALSE, scale = TRUE, ...)
```

Arguments

x	a dispRity object.
center	either a logical value or a numeric vector of length equal to the number of elements of data (default is FALSE).
scale	either a logical value or a numeric vector of length equal to the number of elements of data (default is TRUE).
...	optional arguments to be passed to scale.

Details

To scale or and center using the full distribution (i.e. all the disparity values) or only the distribution within each subsets of bootstraps you can use the optional argument use.all as a logical. By default is use.all = TRUE and uses all the disparity values not only the ones in the subset.

Author(s)

Thomas Guillerme

See Also

dispRity, test.dispRity, scale.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## Scaling the data
summary(scale.dispRity(disparity, scale = TRUE)) # Dividing by the maximum
## Multiplying by 10 (dividing by 0.1)
summary(scale.dispRity(disparity, scale = 0.1))
```

---

select.axes *Selects ordination axes*

Description

Selects the axes required to explain a cumulative threshold amount of variance in an ordination (e.g. > 95%).

Usage

```
select.axes(data, group, threshold = 0.95, inc.threshold = TRUE)
```

Arguments

data	The trait space to analyse. This can be either a "matrix", "prcomp", "princomp" or a "dispRity" object.
group	Optional, either a list of row numbers or names to be used as different groups or a data.frame with the same k elements as in data as rownames. If data is a "dispRity" object that already contains groups, the group argument is recycled.
threshold	The arbitrary threshold amount of variance (by default this is 0.95).
inc.threshold	Logical, whether to output the axes that contain the threshold value (TRUE; default) or not (FALSE). See details.
	, i.e. the axes necessary to include at least the threshold value

Details

If inc.threshold = TRUE, the returned axes are the ones that contains at least the threshold value (e.g. if the threshold is 0.95, all the returned axes contain at least 0.95 of the variance, potentially more). If inc.threshold = FALSE, the returned axes are the ones before reaching this threshold (e.g. the cumulative variance returned is strictly less or equal to 0.95).

Value

A "dispRity", "axes" object that can be printed, summarised and plot through generic print, summary and plot functions. The object is a list containing:

- $dimensions: the maximum number of dimensions selected across all groups;
- $dim.list: the selected dimensions per group;
- $var: the variance per axes per group;
- $scaled.var: the variance scaled variance per axes per group;
- $cumsum.var: the cumulative scaled variance per axes per group;
- $call: a list containing the $threshold value and the $inc.threshold option used.

Author(s)

Thomas Guillerme

See Also

custom.subsets

Examples

```
## Ordinating the USArrests dataset
ordination <- princomp(USArrests, cor = TRUE)
## Which dimensions to select?
(selected <- select.axes(ordination))
## The selected dimensions
selected$dimensions
```

```
## Visualising the results
plot(selected)

## Same but by grouping the data into three groups
states_groups <- list("Group1" = c("Mississippi","North Carolina",
                                   "South Carolina", "Georgia", "Alabama",
                                   "Alaska", "Tennessee", "Louisiana"),
                      "Group2" = c("Florida", "New Mexico", "Michigan",
                                   "Indiana", "Virginia", "Wyoming", "Montana",
                                   "Maine", "Idaho", "New Hampshire", "Iowa"),
                      "Group3" = c("Rhode Island", "New Jersey", "Hawaii",
                                   "Massachusetts"))
(selected <- select.axes(ordination, group = states_groups))
## Note that the required number of axes is now 4 (instead of 3)
plot(selected)

## Loading some example dispRity data
data(demo_data)
## How many axes are required to explain 99% of the variance
## for each group in the Healy et al 2019 data?
(how_many <- select.axes(demo_data$healy, threshold = 0.99))
summary(how_many)
plot(how_many)
```

`set.root.time` *Adds root time to a tree*

Description

Adds or replace root time to a tree by calculating it's root's depth

Usage

```
set.root.time(tree, present = 0)
```

Arguments

`tree`	A `phylo`, `mutiPhylo` or `dispRity` object that contains trees.
`present`	The age of the most recent tip. By default this is set to `0`.

Examples

```
## A random tree with no root.time
my_tree <- rtree(10)
my_tree$root.time # is NULL
## Adding a root time
my_tree <- set.root.time(my_tree)
my_tree$root.time # is not NULL
```

```
## Rewrite the root time with a different present
my_tree <- set.root.time(my_tree, present = 10)
my_tree$root.time # is older
```

`sim.morpho` *Simulates morphological data.*

Description

Generates a morphological matrix using `rTraitDisc` or `gen.seq.HKY` functions.

Usage

```
sim.morpho(
  tree,
  characters,
  states = 1,
  model = "ER",
  rates,
  substitution = c(stats::runif, 2, 2),
  invariant = TRUE,
  verbose = FALSE
)
```

Arguments

`tree`	A phylogenetic tree to use for generating the characters.
`characters`	The number of morphological characters to generate.
`states`	A `numeric` string of probabilities for the number of states for each character (`default = 1`; i.e. 100% binary state characters; see details).
`model`	Either an implemented (`"ER"`, `"HKY"` or `"MIXED"`) or user defined model (see details).
`rates`	A function an its parameters for the rates distribution (see details).
`substitution`	A function an its parameters for the substitutions distribution (see details; `default = c(runif, 2, 2)`).
`invariant`	`logical`, whether to allow any invariant sites (`default = TRUE`).
`verbose`	Whether to be verbose or not (`default = FALSE`).

Details

- The `model` arguments must be either a user's defined function for generating the discrete morphological characters (that takes the states, rates and substitution arguments) or one of the two following:
  - `"ER"` uses the `ape::rTraitDisc` function with the `"ER"` model argument (= Mk model).

  - "HKY" uses the phyclust::gen.seq.HKY function with kappa sampled from the substitution argument, pi = runif(4) (divided by sum(runif(4))), rate.scale sampled from the rates distribution and L being the number of characters and transforms the purines (A, G) into 0 and the pyrimidines (C, T) into 1.
  - "MIXED" randomly uses "ER" or "HKY" for binary characters and "ER" for any character with more than two states.
  - the user defined model must be a function that generates *a single* discrete morphological character and takes one element from at least the following arguments: tree, states, rates, substitution.
- The states argument attributes a number of states to each character by using the given probability vector for each number of states starting from two. For example states = c(0.7, 0.2, 0.1) will generate 70% of characters with two states, 20% of characters with three states and 10% of characters with four states.
- The rates and substitution arguments require a function that outputs a distribution and its optional parameters. For example rates = c(runif, 1, 10) creates a uniform distribution between 1 and 10 for the rates distribution.

Author(s)

Thomas Guillerme

See Also

check.morpho, apply.NA, rTraitDisc, gen.seq.HKY

Examples

```
set.seed(4)
## A random tree with 15 tips
tree <- rcoal(15)
## Setting up the parameters
my_rates = c(rgamma, rate = 10, shape = 5)
my_substitutions = c(runif, 2, 2)

## HKY binary (15*50)
matrixHKY <- sim.morpho(tree, characters = 50, model = "HKY",
     rates = my_rates, substitution = my_substitutions)

## Mk matrix (15*50) (for Mkv models)
matrixMk <- sim.morpho(tree, characters = 50, model = "ER", rates = my_rates)

## Mk invariant matrix (15*50) (for Mk models)
matrixMk <- sim.morpho(tree, characters = 50, model = "ER", rates = my_rates,
     invariant = FALSE)

## MIXED model invariant matrix (15*50)
matrixMixed <- sim.morpho(tree, characters = 50, model = "MIXED",
     rates = my_rates, substitution = my_substitutions,  invariant = FALSE,
     verbose = TRUE)
```

slice.tree *Time slicing a tree.*

Description

Time slicing through a phylogenetic tree.

Usage

```
slice.tree(tree, age, model, FAD, LAD, keep.all.ancestors = FALSE)
```

Arguments

tree	A phylo object with a root.time element.
age	A single numeric value indicating where to perform the slice.
model	One of the following models: "acctran", "deltran", "random", "proximity", "equal.split" or "gradual.split". Is ignored if method = "discrete". See chrono.subsets for the models description.
FAD, LAD	The first and last occurrence data.
keep.all.ancestors	Optional, whether to also include the ancestors of the tree slice (TRUE) or just the ones linking the elements present at the slice (FALSE; default)

Author(s)

Thomas Guillerme

References

Guillerme T. & Cooper N. **2018**. Time for a rethink: time sub-sampling methods in disparity-through-time analyses. Palaeontology. DOI: 10.1111/pala.12364.

See Also

paleotree::timeSliceTree, chrono.subsets.

Examples

```
set.seed(1)
## Generate a random ultrametric tree
tree <- rtree(20)

## Add some node labels
tree$node.label <- letters[1:19]

## Add its root time
tree$root.time <- max(tree.age(tree)$ages)
```

```
## Slice the tree at age 1.5
tree_slice <- slice.tree(tree, age = 1.5, "deltran")

## The slice at age 0.5 but keeping all the ancestors
deep_slice <- slice.tree(tree, age = 0.5, "deltran",
                         keep.all.ancestors = TRUE)

## Visualising the trees
old_par <- par(mfrow = c(2,2))
plot(ladderize(tree), main = "full tree"); axisPhylo()
abline(v =  tree$root.time - 1.5)
plot(ladderize(tree_slice), main = "tree slice"); axisPhylo()
plot(ladderize(deep_slice), main = "slice with ancestors"); axisPhylo()

par(old_par)
```

`slide.nodes` *Stretching a tree*

Description

Stretches a phylogenetic tree at a particular node

Usage

```
slide.nodes(nodes, tree, slide, allow.negative.root = FALSE)
```

Arguments

`nodes`	A list of the ID nodes to slide (`"integer"`) or names (`"character"`). The first node is `ape::Ntip(tree) + 1`, etc.
`tree`	a `"phylo"` object.
`slide`	the sliding value.
`allow.negative.root`	logical, whether to allow negative branch lengths and moving the root node (`TRUE`) or not (`FALSE`; default).

Details

The sliding works by subtracting the slide value to the branch leading to the node and adding it to the descendant branches. Note that the slide value can be negative to slide nodes the other way (up); the only requirement is that the slide does not lead to negative branch length values.

Value

A `"phylo"` object.

Author(s)

Thomas Guillerme

See Also

remove.zero.brlen

Examples

```
set.seed(42)
## Generating a coalescent tree
tree <- rcoal(5)

## Stretching node 8 up and down
tree_slide_up <- slide.nodes(8, tree, slide = 0.075)
tree_slide_down <- slide.nodes(8, tree, slide = -0.075)

## Display the results
par(mfrow = c(3,1))
plot(tree) ; axisPhylo() ; nodelabels()
plot(tree_slide_up) ; axisPhylo() ; nodelabels()
plot(tree_slide_down) ; axisPhylo() ; nodelabels()

## Stretching many nodes
set.seed(42)
tree <- rtree(50)
move_nodes <- c(99, 93, 53, 86, 58, 63, 60, 84)
tree_slided <- slide.nodes(move_nodes, tree, slide = 0.07)

## Display the results
par(mfrow = c(2, 1))
node_colors <- c("lightblue", "orange")[((1:Nnode(tree))+Ntip(tree)) %in% move_nodes + 1]
plot(tree, show.tip.label = FALSE) ; axisPhylo()
nodelabels(bg = node_colors, cex = 0.5)
plot(tree_slided, show.tip.label = FALSE) ; axisPhylo()
nodelabels(bg = node_colors, cex = 0.5)
```

---

sort.dispRity *Sorting or ordering a* dispRity *object.*

Description

Sort (or order) the subsets of a dispRity object.

Usage

```
## S3 method for class 'dispRity'
sort(x, decreasing = FALSE, sort, ...)
```

Arguments

x	A dispRity object.
decreasing	logical. Should the sort be in ascending or descending order? Is ignored if sort is used.
sort	An optional vector of numeric values corresponding to the order in which to return the subsets.
...	optional arguments to be passed to sort.

Author(s)

Thomas Guillerme

See Also

dispRity, test.dispRity, plot.dispRity, get.subsets, get.disparity.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## Sorting the data
summary(disparity)
summary(sort(disparity, decreasing = TRUE))
summary(sort(disparity, sort = c(7,1,3,4,5,2,6)))
```

---

space.maker *Creating multidimensional spaces*

Description

Creates a multidimensional space with a given number of elements and dimensions

Usage

```
space.maker(
  elements,
  dimensions,
  distribution,
  arguments = NULL,
  cor.matrix = NULL,
  scree = NULL,
  elements.names = NULL,
  replicates = NULL
)
```

Arguments

elements	An numeric value.
dimensions	An numeric value smaller than elements.
distribution	One or more functions to determine the distribution of the elements along each dimension. The function must have a single input: elements.
arguments	Optional list of arguments to be passed to the distributions functions in the order they appear (default = NULL, see details).
cor.matrix	An optional correlation matrix of size dimensions * dimensions (default = NULL, see details).
scree	An optional proportional numeric vector for approximating the dimensions variance (default = NULL, see details).
elements.names	Optional, a character or integer string for naming the elements in the matrix.
replicates	Optional, an integer to replicate the simulations and generating multiple spaces.

Details

When passing additional arguments to different distributions, these must be given as a list to each function in the order they appear. For example if distribution = c(runif, rnorm, rgamma) and one wants the distributions to be runif(elements, min = 1, max = 10), rnorm(elements, mean = 8) and rgamma(elements, shape = 1, log = TRUE), the additional arguments should be passed as c(list(min = 1, max = 10), list(mean = 8), list(shape = 1, log = TRUE). If no arguments have to be passed to a certain function, it can be left as NULL (e.g. c(list(min = 1, max = 10), list(NULL), list(shape = 1, log = TRUE)).

The cor.matrix argument should be a correlation matrix between the dimensions. If not NULL, the multidimensional space is multiplied by the the Choleski decomposition (chol) of the correlation matrix. The scree argument is simply a value multiplier for each dimension to adjust their variance to approximate the scree one.

Author(s)

Thomas Guillerme

See Also

null.test, test.dispRity.

Examples

```
## A square space
plot(space.maker(5000, 2, runif), pch = 20)

## A circular space
plot(space.maker(5000, 2, rnorm), pch = 20)

## A 2-dimensional cylindrical space
plot(space.maker(5000, 2, c(rnorm, runif)), pch = 20)
```

```
## A 4-dimensional space with different distributions
space.maker(5, 4, c(runif, runif, rnorm, rgamma),
     arguments = list(list(min = 1, max = 10), list(min = 1, max = 2),
     list(mean = 8), list(shape = 1)))

## A 3-dimensional correlated space
cor_matrix <- matrix(cbind(1, 0.8 ,0.2, 0.8, 1, 0.7, 0.2, 0.7, 1), nrow = 3)
space <- space.maker(10000, 3, rnorm, cor.matrix = cor_matrix)
round(cor(space), 1) ; cor_matrix ## Both should be really similar matrices

## A 3-dimensional space with a priori approximated variance for each dimension
space <- space.maker(10000, 3, rnorm, scree = c(0.6, 0.3, 0.1))
## The resulting screeplot
barplot(apply(space, 2, var))

## Generate 3 2D normal spaces with rownames
space.maker(10, 2, rnorm, elements.names = letters[1:10], replicates = 3)

## Not run:
require(scatterplot3d)
## A cube space
scatterplot3d(space.maker(5000, 3, runif), pch = 20)

## A plane space
scatterplot3d(space.maker(5000, 3, c(runif, runif, runif),
     arguments = list(list(min = 0, max = 0), NULL, NULL)), pch = 20)

## A sphere space
scatterplot3d(space.maker(5000, 3, rnorm), pch = 20)

## A 3D cylindrical space
scatterplot3d(space.maker(5000, 3, c(rnorm, rnorm, runif)), pch = 20)

## Generating a doughnut space
doughnut <- space.maker(5000, 3, c(rnorm, random.circle),
     arguments = list(list(mean = 0), list(runif, inner = 0.5, outer = 1)))
## Reodering the axis for projecting the dougnut in 2D
scatterplot3d(doughnut[,c(2,1,3)], pch = 20)

## End(Not run)
```

`summary.dispRity` *dispRity object summary*

Description

Creates a summary of a `dispRity` object.

Usage

```
## S3 method for class 'dispRity'
summary(
  object,
  ...,
  quantiles = c(50, 95),
  cent.tend = median,
  recall = FALSE,
  digits
)
```

Arguments

object	A dispRity object.
...	Additional arguments to be passed to summary or cent.tend.
quantiles	The quantiles to display (default is quantiles = c(50, 95); is ignored if the dispRity object is not bootstrapped).
cent.tend	A function for summarising the bootstrapped disparity values (default is median).
recall	logical value specifying whether to recall the dispRity parameters input (default = FALSE).
digits	Optional, a value for digits the values in the output table (default = 2).

Details

If the dispRity object to summarise comes from a chrono.subsets using a "multiPhylo" object, the displayed number of observations (n) corresponds to the maximum number of observation at the specific time slice (some slices through some trees might have less observations).

Value

A data.frame with:

subsets	the subset names.
n	the maximum number of elements in each subset (see details).
observed	the observed disparity or the the observed central tendency (<cent_tend>) of disparity (obs.<cent_tend>).
bootstraps...	if data is bootstrapped, the bootstrapped disparity's central tendency (bs.<cent_tend>) and the quantiles of the bootstrapped disparities (or, if data is not bootstrapped but disparity is calculated as a distribution - see dispRity) - the quantiles of the observed disparity are displayed).

Author(s)

Thomas Guillerme

See Also

dispRity, plot.dispRity.

Examples

```
## Load the disparity data based on Beck & Lee 2014
data(disparity)

## Summarising the results
summary(disparity) # default
## Using different options
summary(disparity, quantiles = 75, cent.tend = mean, digits = 8,
     recall = TRUE)
```

`test.dispRity` *Testing disparity hypotheses*

Description

Applying statistical tests to dispRity objects

Usage

```
test.dispRity(
  data,
  test,
  comparisons = "pairwise",
  rarefaction = NULL,
  correction = "none",
  concatenate = TRUE,
  conc.quantiles = c(mean, c(95, 50)),
  details = FALSE,
  ...
)
```

Arguments

`data`	A `dispRity` object.
`test`	A test `function` to apply to the data.
`comparisons`	If data contains more than two subsets, the type of comparisons to apply: either `"pairwise"` (default), `"referential"`, `"sequential"`, `"all"` or a list of pairs of subset names/number to compare (see details).
`rarefaction`	A `numeric` value indicating whether to use a specific rarefaction level (default = `NULL`).
`correction`	Which p-value correction to apply to `htest` category test (see `p.adjust`; default = `"none"`).
`concatenate`	Logical, whether to concatenate bootstrapped disparity values (`TRUE`; default) or to apply the test to each bootstrapped value individually (`FALSE`).

conc.quantiles	If concatenate = TRUE, must be a central tendency function and a vector of quantiles (default = c(mean, c(95, 50))).
details	Whether to output the details of each test (non-formatted; default = FALSE).
...	Additional options to pass to the test function.

Details

The comparison argument can be:

- "pairwise": pairwise comparisons of all the subsets (default).
- "referential": compares the first subset to all the others.
- "sequential": compares each subset sequentially (e.g. first against second, second against third, etc.).
- "all": compares all the subsets simultaneously to the data (i.e. bootstrapped disparity ~ subsets names). This argument is used for lm or glm type tests.
- A list of pairs of number of subsets to compare. Each element of the list must contain two elements (e.g. list(c("a","b"), ("b", "a")) to compare "a" to "b" and then "b" to "a").
- If the called test is null.test, the comparison argument is ignored.

IMPORTANT: if you are performing multiple comparisons (e.g. when using "pairwise", "referential" or "sequential"), don't forget about the Type I error rate inflation. You might want to use a *p-value* correction (see p.adjust).

Author(s)

Thomas Guillerme

See Also

dispRity, null.test, bhatt.coeff, pair.plot, adonis.dispRity, randtest.dispRity, test.dispRity

Examples

```
## Load the Beck & Lee 2014 data
data(BeckLee_mat50)
data(BeckLee_tree)

## Calculating the disparity from customised subsets
## Generating the subsets
groups <- crown.stem(BeckLee_tree, inc.nodes = FALSE)
customised_subsets <- custom.subsets(BeckLee_mat50, groups)
## Bootstrapping the data
bootstrapped_data <- boot.matrix(customised_subsets, bootstraps = 100)
## Calculating the sum of variances
sum_of_variances <- dispRity(bootstrapped_data, metric = c(sum, variances))

## Measuring the subset overlap
test.dispRity(sum_of_variances, bhatt.coeff, "pairwise")
```

```
## Measuring differences from a reference subset
test.dispRity(sum_of_variances, wilcox.test, "referential")

## Measuring disparity as a distribution
disparity_var <- dispRity(bootstrapped_data, metric = variances)
## Differences between the concatenated bootstrapped values of the subsets
test.dispRity(disparity_var, test = t.test, comparisons = "pairwise",
     concatenate = TRUE, correction = "bonferroni")
## Differences between the subsets bootstrapped
test.dispRity(disparity_var, test = t.test, comparisons = "pairwise",
     concatenate = FALSE, correction = "bonferroni",
     conc.quantiles = c(mean, c(95, 5)))
```

`test.metric` *Test disparity metric*

Description

Test whether a metric captures changes trait space size, density and position.

Usage

```
test.metric(
  data,
  metric,
  ...,
  shifts,
  shift.options,
  model,
  replicates = 3,
  steps = 10,
  dimensions,
  verbose = FALSE,
  save.steps = FALSE
)
```

Arguments

`data`	A matrix or a `dispRity` object (see details).
`metric`	A vector containing one to three functions. At least of must be a dimension-level 1 or 2 function (see details). If `data` is a `dispRity` object with disparity already calculated, this argument can be left empty (and the one from `data` is recycled)
`...`	Optional arguments to be passed to the metric.
`shifts`	The types of shits to test, can be `"random"`, `"size"`, `"density"`, `"evenness"` and `"position"`. See details.
`shift.options`	Optional, a `list` of named arguments to be passed to `reduce.space`

model	Optional, which model to fit for testing the metric. See details.
replicates	A numeric number of replicates to increase variance. By default replicates = 3. If replicates = 1, the model is not run.
steps	The number of steps in the space reduction to output between 10% and 100%. By default steps = 10.
dimensions	Optional, a numeric value or proportion of the dimensions to keep.
verbose	A logical value indicating whether to be verbose (TRUE) or not (FALSE; default).
save.steps	A logical value indicating whether to save the data for visualising the the shift steps if plotting the results (TRUE) or not (FALSE; default).

Details

For the three non-random shifts: "size", "density", "evenness" and "position", the function returns both of shifts as:

- "size.inner" and "size.outer" removing data from the edges or the centre respectively (contracting the size and "hollowing" it respectively).
- "density.higher" and "density.lower" removing data to increase or decrease density respectively (increasing/decreasing nearest neighbour distance).
- "evenness.flattened" and "evenness.compacted" removing data to from the centre of the distribution or from the edges to resepectively "flatten" or "condense" the distribution.
- "position.top" and "position.bottom" removing data from one side or the other of the space (the sides are selected from the point with lowest/highest scores on each dimensions respectively).

See figure 2 in Guillerme et al. 2020 for more details.

The default model is a linear model using the following function: model = function(data) lm(disparity ~ reduction, data) You can provide your own as long as it is a single function with data as a single argument. The two terms from data should be called reduction for the variable on the x axis and disparity for the variable on the y axis. For example: model = function(data) nls(disparity ~ a*reduction/(b+reduction), data) Note that models (like this example) should be specific to the dataset. Any type of model can be fitted but only the ones with an associated summary function will be correctly displayed by summary.dispRity. To not run any model, use model = NULL.

Value

This function outputs a dispRity object containing a list of simulated reductions in trait space. The results can be accessed through the usual S3 methods (print, summary, plot) or accessed directly through x$<name_of_the_shift> (e.g. x$random for the random shift results).

Author(s)

Thomas Guillerme

References

Guillerme T, Puttick MN, Marcy AE, Weisbecker V. **2020** Shifting spaces: Which disparity or dissimilarity measurement best summarize occupancy in multidimensional spaces?. Ecol Evol. 2020;00:1-16. (doi:10.1002/ece3.6452)

See Also

reduce.space dispRity plot.dispRity

Examples

```
## Creating a 2D uniform space
space <- space.maker(300, 2, runif)

## A simple test with only 1 replicate for two shifts (random and size):
simple_test <- test.metric(space, metric = c(prod, ranges),
                           replicates = 1, shifts = c("random", "size"))

## Summarising the tests
summary(simple_test)

## Visualising the test
plot(simple_test)

## Applying the test directly on a disparity object
data(disparity)
median_centroid_test <- test.metric(disparity, shifts = "size")

## Summarising the tests
summary(median_centroid_test)

## Visualising the test
plot(median_centroid_test)

## Not run:
## Note that the tests can take several minutes to run.

## Testing the sum of variance on all shifts
sum_var_test <- test.metric(space, metric = c(sum, variances),
                            shifts = c("random", "size", "density", "position"))

## Summarising the tests
summary(sum_var_test)

## Visualising the test
plot(sum_var_test)

## Creating a 2D uniform space
space <- space.maker(300, 2, runif)

## Re-running the test on two shifts with data saving for visualisation
median_centroid_test <- test.metric(space,
```

```
                                    metric = c(median, centroids),
                                    shifts = c("random", "size"),
                                    save.steps = TRUE)

## Visualising the tests results and display the shifts visualisation
plot(median_centroid_test)

## End(Not run)
```

`tree.age` *Extracting the age of nodes and tips in a tree.*

Description

Extracting the age of each node and tip in a tree give the height of the tree or some specified age.

Usage

```
tree.age(tree, age, order = "past", fossil = TRUE, digits = 4)
```

Arguments

`tree`	A `phylo` object.
`age`	The age of the tree. If missing the age is set to be the tree height.
`order`	Either "past" if the units express time since the present (e.g. million years ago), or "present" if the unit is expressed in time since the root.
`fossil`	`logical`, whether to always consider the tree as containing at least one living taxa (`TRUE`) or allowing only fossil taxa (`FALSE` - default), see details.
`digits`	A `numeric` value or `integer` for the precision of the output.

Details

When `fossil = TRUE`, if the `tree` contains a `tree$root.time` element (for tree's root age), and that `order` is set to `"past"`, the output ages are adjusted to be starting from the root.time. Else, if no `tree$root.time` exists or `fossil = FALSE`, tips and nodes age is relative from the tip furthest away from the root. *THIS FUNCTION DOES NOT ESTIMATE TREE AGES*, it just extracts branch length information and converts it into time units. For basic dating functions in R, check `chronos`, `chronopl`, `chronoMPL` or use more specialised dating software (e.g. MrBayes, BEAST, RAxML, etc.).

Author(s)

Thomas Guillerme

See Also

`slice.tree`, `chrono.subsets`.

Examples

```
## A dated random phylogeny with a root 50 units of time old.
tree.age(rtree(10), age = 50)
## A random tree with the distance since the root.
tree.age(rtree(10), order = 'present')
```

Index