Package 'dispRity'

September 25, 2020

Title Measuring Disparity

Author Thomas Guillerme [aut, cre, cph],
Mark N Puttick [aut, cph]

Maintainer Thomas Guillerme <guillert@tcd.ie>

Version 1.5.0

Date 2020-09-25

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.

Depends R (>= 4.0.0), ape, stats

Imports ade4, castor, Claddis, geiger, geometry, geoscale, graphics, grDevices, methods, mnormt, parallel, phangorn, phyclust, utils, vegan, scales

License GPL-3 | file LICENSE

Suggests testthat, knitr

RoxygenNote 7.1.0

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-09-25 17:20:02 UTC

R topics documented:

`dispRity-package` *Measuring Disparity in R*

Description

A modular package for measuring disparity from multidimensional matrices. 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 basic statistical tests for disparity analysis.

Author(s)

Thomas Guillerme <guillert@tcd.ie>

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

Description

Passing `dispRity` objects to the `adonis` 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 adonis and eventually to vegdist (see details - default method ="euclidean")
...	Any optional arguments to be passed to adonis
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 adonis.

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

See Also

adonis, test.dispRity, custom.subsets, chrono.subsets.

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)
```

`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

Source

https://www.royalsocietypublishing.org/doi/abs/10.1098/rspb.2014.1278

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 *Bhattacharrya 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,
  rarefaction = FALSE,
  dimensions,
  verbose = FALSE,
  boot.type = "full",
  prob
)
```

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`).
`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).
`dimensions`	Optional, a `numeric` value or proportion of the dimensions to keep.
`verbose`	A `logical` value indicating whether to be verbose or not.
`boot.type`	The bootstrap algorithm to use (`default = "full"`; see details).
`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).

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 ordinated matrix with only 3 dimensions
boot.matrix(BeckLee_mat50, bootstraps = 20, dimensions = 3)
## 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)` 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
```

---

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, method, time, model, inc.nodes = FALSE,
                    FADLAD, verbose = FALSE, t0 = FALSE, bind.data = FALSE)
```

Arguments

data	A matrix or a list of matrices.
tree	A phylo or a 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	An optional data.frame 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).

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 is can be one of the following:

- Punctuated models:
  - "acctran": always the value from the ancestral node.
  - "deltran": always the value from the descendant node or tip.
  - "random": randomly selected from the ancestral node or the descendant node or tip.
  - "proximity": selects the ancestral node or the descendant with a probability relative to branch length.
- Gradual models:
  - "equal.split": randomly selected from the ancestral node or the descendant node or tip with a 50% probability each.

- "gradual.split": selects 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.

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

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)
```

Arguments

data	A data.frame or matrix with the elements names as row names.
tree	A phylo or multiPhylo object.

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
```

---

combine.subsets *Combines or cleans subsets.*

Description

Combines multiple subsets together or cleans a subset series to contain at least n elements.

Usage

```
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 numeric value of the minimum of elements for each series (see details).

Details

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).

Value

A `dispRity` object containing the original matrix and subsets. NOTE: if the data are already bootstrapped/rarefied or/and disparity already calculated the operation will have to be performed again.

Author(s)

Thomas Guillerme

See Also

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

Examples

```
## Generate subsets from a dummy matrix
dummy_matrix <- matrix(rnorm(120), 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)
```

`crown.stem` *Separates stem and crown species*

Description

Selects the crown

Usage

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

Arguments

tree	a code"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)
```

Arguments

data	A matrix or a list of matrices.
group	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 group is a phylo object matching data, groups are automatically generated as clades.

Details

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

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

`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)
```

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,
  ...,
  between.groups = FALSE,
  verbose = 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).
dimensions	Optional, a numeric value or 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).
verbose	A logical value indicating whether to be verbose or not.

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:

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.
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.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 eqnkth 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**.
- `ellipse.volume`: calculates the ellipsoid volume of a matrix.
  - WARNING: this function assumes that the input matrix is ordinated and calculates the matrix' eigen values from the matrix as `abs(apply(var(matrix),2,sum))` (which is equivalent to `eigen(var(matrix))$values` but faster). These values are the correct eigen values for any matrix but differ from the ones output from `cmdscale` and `pcoa` because these later have their eigen values multiplied by the number of elements - 1 (i.e. `abs(apply(var(matrix),2,sum)) * nrow(matrix) -1`). Specific eigen values can always be provided manually through `ellipse.volume(matrix,eigen.value = my_val)` (or `dispRity(matrix,metric = ellipse.volume,eigen.value = my_val)`).
- `func.div`: The functional divergence (Vill'eger et al. 2008): the ratio of deviation from the centroid (this is similar to `FD::dbFD()$FDiv`).
- `func.eve`: The functional evenness (Vill'eger 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`).

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 tip and node and their ancestral. This function needs either (1) `matrix`/`list` from `nodes.coordinates`; or a `tree` (`"phylo"`) and `full` (`"logical"`) argument to calculate the node coordinates for the direct descendants (`full = FALSE`) or all descendants down to the root (`full = TRUE`). NOTE: distance is calculated as `"euclidean"` by default, this can be changed using the `method` argument.

- `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 (not 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 (Lalibert'e 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.
- `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`. Extract 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.
- `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,...)` (Foote 1990). 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.
- `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:

- `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 100
- `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.

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.

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 direct ancestral nodes
direct_anc_centroids <- nodes.coordinates(dummy_matrix, rand_tree, full = FALSE)
## Calculating all the ancestral nodes
all_anc_centroids <- nodes.coordinates(dummy_matrix, rand_tree, full = TRUE)
## Calculating the distances from the direct ancestral nodes
ancestral.dist(dummy_matrix, nodes.coords = direct_anc_centroids)
## Calculating the distances from all the ancestral nodes
ancestral.dist(dummy_matrix, nodes.coords = all_anc_centroids)

## 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)

## 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

## 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")

## ellipse.volume
## Ellipsoid volume of a matrix
ellipse.volume(dummy_matrix)
## Calculating the same volume with provided eigen values
ordination <- prcomp(dummy_matrix)
## Calculating the ellipsoid volume
ellipse.volume(ordination$x, eigen.value = ordination$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")

## 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)

## 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))
```

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

Description

A wrapper for the `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.
`nsim`	The number of simulations to calculate null disparity-through-time.
`model`	A evolutionary model for the simulations (see `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 `dtt`.

Details

See dtt for details. Note that for calculating the default metrics implemented in dtt (i.e c("avg.sq","avg.manhattan","nu
this implementation in dispRity is much slower!

Author(s)

Thomas Guillerme

See Also

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

Examples

```
## Loading geiger's example data set
require(geiger)
geiger_data <- get(data(geospiza))

## Calculate the disparity of the dataset using dtt::geiger
geiger_dtt <- dtt(phy = geiger_data$phy, data = geiger_data$dat, nsim = 100)

## 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 = geiger_data$dat, metric = average.sq,
                             tree = geiger_data$phy, nsim = 100)

## Plotting the results
plot(dispRity_dtt)

## Disparity values are identical up to the 9th digit!
round(geiger_dtt$dtt, 9) == round(dispRity_dtt$dtt, 9)

## Calculate disparity with a different metric using dtt.dispRity
dispRity_dtt2 <- dtt.dispRity(data = geiger_data$dat, tree = geiger_data$phy,
                             metric = c(median, centroids), nsim = 50)
plot(dispRity_dtt2)
```

---

extinction.subsets *Getting the time subsets from at 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")
```

---

extract.dispRity *Extracting disparity values.*

Description

Extracts the disparity from a dispRity object.

Usage

```
extract.dispRity(
  data,
  subsets,
  observed = TRUE,
  rarefaction = FALSE,
  concatenate = TRUE
)
```

Arguments

data	A dispRity object containing disparity results.
subsets	Optional, a numeric or character for which subsets to get (if missing, the value for all subsets are given).
observed	A logical value indicating whether to output the observed (TRUE (default)) or the bootstrapped values (FALSE).
rarefaction	Optional, a single numeric value corresponding to the rarefaction level (as the number of elements; if missing, the non-rarefied values are output).
concatenate	When the disparity metric is a distribution, whether to concatenate it (TRUE; default) or to return each individual metric.

Author(s)

Thomas Guillerme

See Also

dispRity, get.subsets.

Examples

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

## Extracting the observed disparity
extract.dispRity(disparity)

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

---

fill.dispRity *Fills a* dispRity *object.*

Description

Fills a dispRity object using the data from its matrix

Usage

```
fill.dispRity(data)
```

Arguments

data	A dispRity object.

Author(s)

Thomas Guillerme

Examples

```
## 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)
```

---

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 gpagen or a 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

gpagen, 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.ancestors *Get ancestors*

Description

Gets the list of ancestors (parents) from a tip or a node (modified from getParent)

Usage

```
get.ancestors(tip, tree, full = TRUE)
```

Arguments

tip	A tip (or node) index.
tree	A tree topology of class "phylo".
full	Whether to output the direct ancestor only (FALSE) or the full list of ancestors to the root (TRUE - default)

Value

A integer vector of ancestor(s).

Author(s)

Thomas Guillerme

See Also

ancestral.dist, nodes.coordinates, getParent

Examples

```
## A random tree
tree <- rtree(10)
## Get the ancestors of the first tip
get.ancestors(1, tree)
```

---

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.subsets` *Extracts subsets from a dispRity object.*

Description

Extracting some subsets and data from a `dispRity` object.

Usage

```
get.subsets(data, subsets)
```

Arguments

`data`	A `dispRity` object.
`subsets`	A list of subset names or subset numbers to be extracted.

Value

This function outputs a `dispRity` object.

Author(s)

Thomas Guillerme

See Also

`dispRity`, `extract.dispRity`.

Examples

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

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

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

make.dispRity *Creates a* dispRity *object.*

Description

Creating an empty dispRity object from a matrix

Usage

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

Arguments

data	A matrix.
call	Optional, a list to be a dispRity call.
subsets	Optional, a list to be a dispRity subsets list.

Author(s)

Thomas Guillerme

Examples

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

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

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)
```

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.

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)
```

---

matrix.dispRity *Fetching a matrix from a* dispRity *object.*

Description

Fetching a specific matrix from a dispRity object.

Usage

```
matrix.dispRity(data, subsets, rarefaction, bootstrap, matrix = 1)
```

Arguments

data	A dispRity object.
subsets	Optional, a numeric value to select subsets.
rarefaction	Optional, a numeric value to select the rarefaction level (0 is no rarefaction).
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).

Author(s)

Thomas Guillerme

Examples

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

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

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

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

---

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

```
## 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 = 1000, 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 = 1000, 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")
```

---

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 = "ER",
  threshold = TRUE,
  special.tokens,
  special.behaviours,
  brlen.multiplier,
  verbose = FALSE,
  parallel = FALSE,
  output,
  castor.options
)
```

Arguments

`data`	A `matrix` 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 `vector` of models to be passed to `asr_mk_model`.

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	logical, whether to use parallel algorithm (TRUE) or not (FALSE - default).
output	optional, see Return section below.
castor.options	optional, a named list of options to be passed to function called by asr_mk_model.

Details

The models argument can be a single or a list of transition matrix, a single or a a vector of built-in model(s) (see below) or a list of both matrices and built-in models: The available built-in models in 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 asr_mk_model for more models.

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 `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"`.

Author(s)

Thomas Guillerme

See Also

`asr_mk_model`, `char.diff`

Examples

```
set.seed(42)
## A simple example:
## A random tree with 15 tips
tree <- rcoal(15)
## Setting up the parameters
my_rates = c(rgamma, rate = 10, shape = 5)

## A random Mk matrix (15*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 10 trees
multiple_trees <- rmtree(10, 15)

## 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(28)^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)
```

`nodes.coordinates` *Nodes coordinates*

Description

Calculates ancestral nodes coordinates in a format that can be passed to `ancestral.dist`

Usage

```
nodes.coordinates(matrix, tree, full = TRUE)
```

Arguments

`matrix`	The `matrix` on which `centroids` will be applied
`tree`	A tree topology of class `"phylo"`.
`full`	Whether to get the centroids for all ancestors down to the root (`TRUE` - default) or only the direct ancestors (`FALSE`)

Value

A `matrix` if `full = FALSE` or a `list` of matrices if `full = TRUE`.

Author(s)

Thomas Guillerme

See Also

`ancestral.dist`, `dispRity.metric`, `dispRity`, `get.ancestors`

Examples

```
## A random matrix
matrix <- matrix(rnorm(90), 9, 10)
## A random treee with node labels
tree <- rtree(5) ; tree$node.label <- paste0("n", 1:4)
## Adding the tip and node names to the matris
rownames(matrix) <- c(tree$tip.label, tree$node.label)

## Calculating the direct ancestral nodes
direct_anc_centroids <- nodes.coordinates(matrix, tree, full = FALSE)
## Calculating all the ancestral nodes
all_anc_centroids <- nodes.coordinates(matrix, tree, full = TRUE)
```

```
## Calculating the distances from the direct ancestral nodes
ancestral.dist(matrix, nodes.coords = direct_anc_centroids)
## Calculating the distances from all the ancestral nodes
ancestral.dist(matrix, nodes.coords = all_anc_centroids)
```

`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 = ellipse.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)
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)
```

---

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`.
`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

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 default is `specific.args = list(dimensions = c(1,2), matrix = 1)` where `dimensions` designates which dimensions to plot and `matrix` which specific matrix from `data` to plot.
- for plots with legends (if `type = "preview"`; if data is `"randtest"` or `"test.metric"`) you can pass any non ambiguous arguments for `legend` such as `specific.args = list(legend = list(x = 1, y = 2, bg = "blue"))`. When the plot generates two legends (e.g. when the data is `"test.metric"`), these arguments can be a list (e.g. `specific.args = list(legend = list(list(x = "bottomright"), list(x = "topright")))`. *HINT*: to remove the legends all together you can use `specific.args = list(legend = FALSE)`.

Author(s)

Thomas Guillerme

See Also

`dispRity`, `summary.dispRity`.

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))
```

`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)
```

`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).

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 <- dispRity::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 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" 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 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")
```

`rescale.dispRity` *Rescaling and centering disparity results.*

Description

Scales or/and centers the disparity measurements.

Usage

```
## S3 method for class 'dispRity'
rescale(data, center = FALSE, scale = TRUE, use.all = TRUE, ...)
```

Arguments

`data`	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`).
`use.all`	`logical`, whether to scale/center using the full distribution (i.e. all the disparity values) or only the distribution within each subsets of bootstraps (default is `TRUE`).
`...`	optional arguments to be passed to `scale`.

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

`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)
```

size.subsets *Size subsets.*

Description

Getting the size (number of elements) from each subsets of a dispRity object.

Usage

```
size.subsets(data)
```

Arguments

data	A dispRity object.

Author(s)

Thomas Guillerme

See Also

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

Examples

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

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

slice.tree *Time slicing a tree.*

Description

Time slicing through a phylogenetic tree.

Usage

```
slice.tree(tree, age, model, FAD, LAD)
```

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.

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 <- rcoal(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 0.75
tree_75 <- slice.tree(tree, age = 0.75, "deltran")
```

---

slide.nodes *Stretching a tree*

Description

Stretches a phylogenetic tree at a particular node

Usage

```
slide.nodes(nodes, tree, slide)
```

Arguments

nodes	A list of the ID nodes to slide ("integer"). The first node is ape::Ntip(tree) + 1, etc.
tree	a "phylo" object.
slide	the sliding value.

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`, `extract.dispRity`.

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
)
```

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).

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))

## 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.

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
)
```

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 or not.

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.bottom" and "position.top" 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

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)

## End(Not run)
```

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

Description

Calculates 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 = 3)
```

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.

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