

# Package ‘dispRity’

September 2, 2019

**Title** Measuring Disparity

**Author** Thomas Guillaume [aut, cre, cph],  
Mark N Puttick [aut, cph]

**Maintainer** Thomas Guillaume <guillert@tcd.ie>

**Version** 1.3.1

**Date** 2019-09-02

**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 (Guillaume (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 - Guillaume and Cooper (2018) <doi:10.1111/pala.12364>) or per groups as well as visualising the results. Finally, this package provides several basic statistical tests for disparity analysis.

**Depends** R (>= 3.0.0), ape, stats

**Imports** ade4, Claddis, geiger, geometry, geoscale, graphics,  
grDevices, methods, mnormt, paleotree, phangorn, phyclus,  
utils, vegan

**License** GPL-3 | file LICENSE

**Suggests** testthat, knitr

**RoxygenNote** 6.1.1

**URL** <https://github.com/TGuillaume/dispRity>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2019-09-02 13:20:02 UTC

## R topics documented:

dispRity-package	3
adonis.dispRity	3
apply.NA	5
BeckLee	7

BeckLee_disparity	7
bhatt.coeff	8
boot.matrix	9
char.diff	11
check.morpho	12
chrono.subsets	13
Claddis.ordination	16
clean.data	17
combine.subsets	18
crown.stem	19
custom.subsets	20
disparity	22
dispRity	23
dispRity.metric	25
dispRity.per.group	30
dispRity.through.time	31
dt.dispRity	32
extinction.subsets	33
extract.dispRity	34
fill.dispRity	35
geomorph.ordination	36
get.ancestors	37
get.bin.ages	38
get.contrast.matrix	39
get.subsets	40
make.dispRity	41
make.metric	42
matrix.dispRity	43
model.test	44
model.test.sim	47
model.test.wrapper	50
nodes.coordinates	52
null.test	53
pair.plot	55
plot.char.diff	56
plot.dispRity	57
print.dispRity	60
random.circle	61
reduce.matrix	62
remove.zero.brlen	63
rescale.dispRity	64
sim.morpho	65
size.subsets	67
slice.tree	68
slide.nodes	69
sort.dispRity	70
space.maker	71
summary.dispRity	73

<i>dispRity</i> -package	3
test.dispRity . . . . .	74
tree.age . . . . .	76
<b>Index</b>	<b>78</b>

dispRity-package	<i>Measuring Disparity in R</i>
------------------	---------------------------------

## 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	<i>adonis dispRity (from vegan::adonis)</i>
-----------------	---

## Description

Passing *dispRity* objects to the [adonis](#) function from the *vegan* package.

## Usage

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

## Arguments

data	A <i>dispRity</i> object with subsets
formula	The model formula (default is <code>matrix ~ group</code> , see details)
method	The distance method to be passed to <a href="#">adonis</a> and eventually to <a href="#">vegdist</a> (see details - default method = "euclidean")
...	Any optional arguments to be passed to <a href="#">adonis</a>
warn	logical, whether to print internal warnings (TRUE; default - advised) or not (FALSE).

## 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` 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.
...	Any additional arguments.

## 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, bootstrapped 100 times from the Beck and Lee data

**Source**

<http://rspb.royalsocietypublishing.org/content/281/1793/20141278.short>

**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 <a href="#">bw.nrd0</a> ).
...	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 <a href="#">chrono.subsets</a> or <a href="#">custom.subsets</a> - 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 sample. 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 give 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:

<code>matrix</code>	the multidimensional space (a matrix).
<code>call</code>	A list containing the called arguments.
<code>subsets</code>	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)
```

**Arguments**

matrix            A discrete matrix or a list containing discrete characters.

**Details**

The character difference is calculated as half the sum of the Gower distances between the characters.

**Value**

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

**Author(s)**

Thomas Guillaume

**References**

Gower, J.C. 1966. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika* 53:325-338.

**See Also**

[plot.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)
char.diff(morpho_matrix)
```

---

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 <a href="#">optim.parsimony</a> or a parsimony function that can take a <a href="#">phyDat</a> object as an input (default = "fitch").
first.tree	A list of functions to generate the first most parsimonious tree (default = c( <a href="#">dist.hamming</a> , <a href="#">NJ</a> ); see details).
distance	Optional, if <a href="#">orig.tree</a> is provided, the function to use for measuring distance between the trees (default = <code>link[phangorn]{RF.dist}</code> ).

...	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 Guillaume

### 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)
```

---

chronosubsets

*Separating data in chronological subsets.*

---

### Description

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

**Usage**

```
chronosubsets(data, tree, method, time, model, inc.nodes = FALSE,
              FADLAD, verbose = FALSE, t0 = FALSE)
```

**Arguments**

data	A matrix (see details).
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: "acctrans", "deltrans", "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).

**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:
  - "acctrans": always the value from the ancestral node.
  - "deltrans": 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:

<code>matrix</code>	the multidimensional space (a <code>matrix</code> ).
<code>call</code>	A list containing the called arguments.
<code>subsets</code>	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
chronosubsets(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
chronosubsets(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
chronosubsets(data = BeckLee_mat99, tree = BeckLee_tree,
              method = "continuous", model = "acctrans", 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 <a href="#">ReadMorphNexus</a> or the path to a file to be read by <a href="#">read.nexus.data</a> (see details).
distance	Distance type to be computed by <a href="#">MorphDistMatrix</a> . Can be either "GC", "GED", "RED", "MORD". distance can also be set to NULL to convert a matrix in <a href="#">ReadMorphNexus</a> list type (see details).
...	Any optional arguments to be passed to <a href="#">MorphDistMatrix</a> .
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 <a href="#">cmdscale</a> ) or not (add = FALSE).
arg.cmdscale	Any optional arguments to be passed to <a href="#">cmdscale</a> (as a named list such as <code>list(x.ret = TRUE)</code> ).

### 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 [ReadMorphNexus](#) type list treating all characters as unordered. If the distance is set to NULL, data will be only converted into a [ReadMorphNexus](#) type list.

### Author(s)

Thomas Guillerme

### See Also

[MorphDistMatrix](#), [ReadMorphNexus](#), [MakeMorphMatrix](#), [cmdscale](#), [custom.subsets](#), [chrono.subsets](#), [boot.matrix](#), [dispRity](#).

**Examples**

```

## Not run:
require(Claddis)

## Ordinating the distance matrix of Claddis example data
Claddis.ordination(Claddis::Michaux1989)

## 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 <code>dispart</code> 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.  
 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:

<code>matrix</code>	the multidimensional space (a matrix).
<code>call</code>	A list containing the called arguments.
<code>subsets</code>	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	<i>disparity</i>
-----------	------------------

---

### 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	<i>Calculates disparity from a matrix.</i>
----------	--

---

## Description

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

## Usage

```
dispRity(data, metric, 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).
dimensions	Optional, a numeric value or proportion of the dimensions to keep.
...	Optional arguments to be passed to the metric.
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).

**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, ...)
```

## 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 <a href="#">vegdist</a> (e.g. method = "euclidean" - default - or method = "manhattan") or k.root to scale the result using the eqn <sup>k</sup> th root. See details below for available optional arguments for each function.

## Details

These are inbuilt functions for calculating disparity. See [make.metric](#) for details on `dimension.level3.fun`, `dimension.level2.fun` and `dimension.level1.fun`.

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))$values` but faster). These values are the correct eigen values for any matrix but differ from the ones output from `cmdscale` and `pcoa` because these later have their eigen values multiplied by the number of elements - 1 (i.e. `abs(apply(var(matrix),2,sum)) * nrow(matrix) - 1`). Specific eigen values can always be provided manually through `ellipse.volume(matrix,eigen.value = my_val)` (or `dispRity(matrix,metric = ellipse.volume,eigen.value = my_val)`).
- `func.div`: The functional divergence (Vill'eger et al. 2008): the ratio of deviation from the centroid (this is similar to `dbFD$FDiv`).
- `func.eve`: The functional evenness (Vill'eger et al. 2008): the minimal spanning tree distances evenness (this is similar to `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.
- `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.
- `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)
- `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 *k*th 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 [spanntree](#)). This function can get slow with big matrices. To speed it up, one can directly use distance matrices as the multidimensional space.

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 matrix
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)

## 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)

## 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")

## 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)

## 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")

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

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

## 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)

## 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 <a href="#">dispRity</a> for details).
...	Optional arguments to be passed to <a href="#">custom.subsets</a> , <a href="#">boot.matrix</a> and <a href="#">dispRity</a> .

### 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 [dispRity.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 <a href="#">dispRity</a> for details).
...	Optional arguments to be passed to <a href="#">chrono.subsets</a> , <a href="#">boot.matrix</a> and <a href="#">dispRity</a> .

**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 `dispRity.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))
```

---

<code>dtt.dispRity</code>	<i>dtt dispRity</i> (from <code>geiger::dtt</code> )
---------------------------	--

---

**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", ...)
```

**Arguments**

<code>data</code>	A dispRity object or a matrix
<code>metric</code>	The disparity metric to be passed to <code>dispRity</code> .
<code>tree</code>	A phylo object matching the data and with a <code>root.time</code> element.
<code>nsim</code>	The number of simulations to calculate null disparity-through-time.
<code>model</code>	A evolutionary model for the simulations (see <code>sim.char</code> - default is "BM").
<code>alternative</code>	The H1 alternative (for calculating the p-value). Can be "two-sided" (default), "greater" or "lesser"; see details.
<code>...</code>	Any other arguments to be passed to <code>dtt</code> .

**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 <a href="#">test.dispRity</a> (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	<i>Fills a dispRity object.</i>
---------------	---------------------------------

---

**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 ordines 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 Guillaume

**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 Guillaume

**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	<i>Creates a dispRity object.</i>
---------------	-----------------------------------

---

**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)
```

**Arguments**

<code>fun</code>	Your very own function!
<code>...</code>	Some arguments to be passed to <code>fun</code> .
<code>silent</code>	logical; if <code>FALSE</code> (default), the function will be verbose and give no output; if <code>TRUE</code> , the function will only output the function's dimension-level.

**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](#)).

**Author(s)**

Thomas Guillerme

**See Also**

[dispRity](#), [dispRity.metric](#).

**Examples**

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

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

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

---

matrix.dispRity	<i>Fetching a matrix from a dispRity object.</i>
-----------------	--

---

**Description**

Fetching a specific matrix from a dispRity object.

**Usage**

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

**Arguments**

data	A dispRity object.
subsets	A numeric value to select subsets (0 is no subsets; default).
rarefaction	A numeric value to select the rarefaction level (0 is no rarefaction; default).
bootstrap	A numeric value to select a specific bootstrap draw (0 is no bootstrap; default).

**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 second subset
matrix.dispRity(disparity, subsets = 2)

## 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 <code>dispRity</code> 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 heterogenous model, either using a single input or a list containing different models (See <b>Details</b> ). If a vector with multiple modes is supplied then the model will test for shifts in modes at the time supplied by <code>time.split</code> .
pool.variance	If <code>NULL</code> (default) the difference in variances will be calculated using <code>bartlett.test</code> 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 <code>TRUE</code> or <code>FALSE</code> 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 ( <code>numeric</code> ). 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 <b>Details</b> ).
fixed.optima	A logical value, whether to use an estimated optimum value in OU models ( <code>FALSE</code> - default), or whether to set the OU optimum to the ancestral value ( <code>TRUE</code> ).
control.list	A list of fine-tune control inputs for the <code>optim</code> function.
verbose	logical, whether to display the model results while they are computed ( <code>TRUE</code> - default).

## Details

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 heterogenous model (see list in <a href="#">model.test</a> ) or (ii) an object of class <code>disprity</code> returned from <code>model.test</code> function. If a <code>disprity</code> object is supplied, all remaining arguments apart from <code>sim</code> and <code>model.rank</code> and <code>alternative</code> are ignored as the model specified by the input model is used.
model.rank	If a <code>disprity</code> object is supplied, which model is used for simulation. The rank refers to the order of models as specified by <code>AICc</code> , so if <code>model.rank = 1</code> (default) the best-fitting model is used for simulation.
alternative	If the simulation is based on a <code>disprity</code> 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 <code>time.span</code> 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 <code>time.span</code> 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

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

<code>data</code>	A <code>disprity</code> object used to test models of evolution through time.
<code>model</code>	The model(s) of evolution to allow for changes in disparity-through-time using a homogenous or heterogenous model, either using a single input or a list containing different models (see list in <code>model.test</code> ). If a vector with multiple modes is supplied then the model will test for shifts in modes at the time supplied by <code>time.split</code> .
<code>pool.variance</code>	If <code>NULL</code> (default) the difference in variances will be calculated using <code>bartlett.test</code> 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 <code>TRUE</code> or <code>FALSE</code> are used, Bartlett's test will be ignored and the analyses will use the user-set pooling of variances.
<code>time.split</code>	The age of the change in mode ( <code>numeric</code> ). 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 <b>Details</b> ).
<code>fixed.optima</code>	A logical value, whether to use an estimated optimum value in OU models ( <code>FALSE</code> - default), or whether to set the OU optimum to the ancestral value ( <code>TRUE</code> ).
<code>control.list</code>	A list of fine-tune control inputs for the <code>optim</code> function.
<code>verbose</code>	logical, whether to display the model results as computed ( <code>TRUE</code> - default).
<code>sim</code>	The number of separate simulations (default = 1000).
<code>plot.sim</code>	Logical. If <code>TRUE</code> (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 <a href="#">plot.dispRity</a> for more details. If missing, the default colours <code>c("black", "lightgrey", "grey")</code> 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 <code>plot.sim = TRUE</code> , 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 <code>show.p = TRUE</code> ). If missing, the value is set to 1.
legend	Logical, when <code>plot.sim = TRUE</code> , whether to display the legend in the first panel (TRUE) or not (FALSE - default).
...	Any additional arguments to be passed to <a href="#">plot.dispRity</a> or <a href="#">summary.dispRity</a> .

### 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)`.

### 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 = 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)
```

---

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 <a href="#">centroids</a> 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 tree with node labels
tree <- rtree(5) ; tree$node.label <- paste0("n", 1:4)
## Adding the tip and node names to the matrix
rownames(matrix) <- c(tree$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**

<code>data</code>	a <code>dispRity</code> object.
<code>replicates</code>	the number of replicates for the test (default = 100).
<code>null.distrib</code>	one or more distribution functions to generate the null model to be passed to <a href="#">space.maker</a> .
<code>null.args</code>	any additional distribution arguments to be passed to <a href="#">space.maker</a> (see arguments within; default = NULL).

null.cor	an additional correlation matrix to be passed to <a href="#">space.maker</a> (see <code>cor.matrix</code> within; default = NULL).
null.screen	an additional vector of variance per axis (equivalent to <a href="#">screplot</a> output); default = NULL).
alter	the type of alternative hypothesis (H1) as used in <a href="#">randtest</a> (default = "two-sided").
scale	whether to scale the simulated and the observed data.
...	optional arguments to be passed to <a href="#">as.randtest</a> .

**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	<i>Plots pairwise comparisons</i>
-----------	-----------------------------------

---

**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 <a href="#">plot</a> .

**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 <code>char.diff</code> .
...	Any additional graphical arguments to be passed to <code>image</code> .
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, ylim,
     xlab, ylab, col, chrono.subsets = TRUE, observed = FALSE,
     add = FALSE, density = NULL, element.pch = 15, dimensions = c(1,
     2), nclass = 10, coeff = 1)
```

**Arguments**

x	A dispRity object.
...	Any optional arguments to be passed to <a href="#">plot</a> .
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 <a href="#">chrono.subsets</a> and "box" for <a href="#">custom.subsets</a> . See details.
quantiles	The quantiles to display (default is <code>quantiles = c(50,95)</code> ); is ignored if the dispRity object is not bootstrapped).
cent.tend	A function for summarising the bootstrapped disparity values (default is <a href="#">median</a> ).
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.
ylim	Optional, two numeric values for the range of the y axis.
xlab	Optional, a character string for the caption of the x axis.
ylab	Optional, one or two (if <code>elements = TRUE</code> ) character string(s) for the caption of the y axis.
col	Optional, some character string(s) for the colour of the plot.
chrono.subsets	logical whether to handle continuous data from the <code>chrono.subsets</code> function as time (in Ma). When this option is set to TRUE for other type options, the names of the subsets are used for the x axis labels.
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 <a href="#">polygon</a> . Is ignored if <code>type = "box"</code> or <code>type = "line"</code> .
element.pch	optional, if <code>elements = TRUE</code> , the point type to represent them (default are squares: <code>element.pch = 15</code> ).
dimensions	optional, if <code>type = "preview"</code> , a pair of "numeric" values of which dimensions to display (default is <code>c(1,2)</code> ).
nclass	when plotting a <a href="#">null.test</a> the number of nclass argument to be passed to <a href="#">hist</a> (default = 10).
coeff	when plotting a <a href="#">null.test</a> the coefficient for the magnitude of the graph (see <a href="#">randtest</a> ; default = 1).

## 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!

## Author(s)

Thomas Guillerme

## See Also

[dispRity](#), [summary.dispRity](#), [pair.plot](#).

## 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, 5),
      xlab = ("Time (Ma)", ylab = "disparity")

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

## Using different options (with non time.slicing option)
plot(disparity, type = "continuous", chrono.subsets = FALSE,
      elements = TRUE, col = c("red", "orange", "yellow"))

## 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))

## Not run:
## Geoscale plots
require(geoscale)

## Converting the data into a list
data_obs <- extract.dispRity(disparity, observed = TRUE)
data_distribution <- extract.dispRity(disparity, observed = FALSE)
## Removing one list level
data_distribution <- unlist(data_distribution, recursive = FALSE)
data_obs <- as.vector(data_obs)

## Getting the ages
ages <- as.numeric(names(disparity$subsets))

## Plotting the results median
geoscalePlot(ages, data_obs, boxes = "Age", data.lim = c(1.5, 2), type = "l")

## Plotting the results distribution
geoscaleBox(data_distribution, ages, boxes = "Age", data.lim = c(1.5, 2))

## End(Not run)

```

---

`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

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

## 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	<i>Random circle</i>
---------------	----------------------

---

**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	<i>Reduce a matrix</i>
---------------	------------------------

---

**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 <a href="#">vegdist</a> , 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)
```

---

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	<i>Size subsets.</i>
--------------	----------------------

---

## 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 <a href="#">chrono.subsets</a> 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

[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**

<code>nodes</code>	A list of the ID nodes to slide ("integer"). The first node is <code>ape::Ntip(tree) + 1</code> , etc.
<code>tree</code>	a "phylo" object.
<code>slide</code>	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	<i>Sorting or ordering a dispRity object.</i>
---------------	---

---

## 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 Guillaume

**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)))
## Reordering the axis for projecting the doughnut 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 <a href="#">summary</a> .
quantiles	The quantiles to display (default is <code>quantiles = c(50, 95)</code> ); is ignored if the dispRity object is not bootstrapped).
cent.tend	A function for summarising the bootstrapped disparity values (default is <a href="#">median</a> ).
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 <a href="#">dispRity</a> ) - 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	<i>Testing disparity hypotheses</i>
---------------	-------------------------------------

---

**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 <a href="#">p.adjust</a> ; 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 Guillaume

## 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)))
```

---

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

adonis, [3](#), [4](#)  
adonis.dispRity, [3](#), [75](#)  
ancestral.dist, [38](#), [52](#), [53](#)  
ancestral.dist (dispRity.metric), [25](#)  
apply.NA, [5](#), [66](#)  
as.randtest, [54](#)

bartlett.test, [44](#), [50](#)  
BeckLee, [7](#), [8](#), [22](#)  
BeckLee\_ages (BeckLee), [7](#)  
BeckLee\_disparity, [7](#)  
BeckLee\_mat50 (BeckLee), [7](#)  
BeckLee\_mat99 (BeckLee), [7](#)  
BeckLee\_tree (BeckLee), [7](#)  
bhatt.coeff, [8](#), [75](#)  
boot.matrix, [8](#), [9](#), [15](#), [16](#), [19](#), [21–24](#), [30–32](#),  
[37](#), [61](#), [67](#)  
bw.nrd0, [8](#)

centroids, [22](#), [52](#)  
centroids (dispRity.metric), [25](#)  
char.diff, [11](#), [57](#)  
check.morpho, [12](#), [39](#), [66](#)  
chol, [71](#)  
chrono.subsets, [4](#), [8–10](#), [13](#), [16](#), [19](#), [21–24](#),  
[31–34](#), [37](#), [39](#), [58](#), [61](#), [67](#), [68](#), [73](#), [77](#)  
CI, [13](#)  
Claddis.ordination, [16](#)  
clean.data, [17](#)  
cmdscale, [16](#), [26](#)  
combine.subsets, [18](#)  
convhull.surface (dispRity.metric), [25](#)  
convhull.volume (dispRity.metric), [25](#)  
convhulln, [25](#)  
crown.stem, [19](#), [21](#)  
cust.series (custom.subsets), [20](#)  
cust.subsets, [10](#), [15](#)  
cust.subsets (custom.subsets), [20](#)  
custom.series (custom.subsets), [20](#)  
custom.subsets, [4](#), [9](#), [16](#), [19](#), [20](#), [20](#), [23](#), [24](#),  
[30](#), [33](#), [36](#), [37](#), [58](#), [61](#), [67](#)

dbFD, [26](#)  
diagonal (dispRity.metric), [25](#)  
dimension.level1.fun (dispRity.metric),  
[25](#)  
dimension.level2.fun (dispRity.metric),  
[25](#)  
dimension.level3.fun (dispRity.metric),  
[25](#)  
disparity, [22](#)  
displacements (dispRity.metric), [25](#)  
dispRity, [8](#), [10](#), [15](#), [16](#), [19](#), [21](#), [22](#), [23](#), [27](#),  
[30–32](#), [35](#), [37](#), [40](#), [42](#), [53](#), [59](#), [61](#), [65](#),  
[67](#), [70](#), [73–75](#)  
dispRity-package, [3](#)  
dispRity.metric, [23](#), [24](#), [25](#), [30–32](#), [42](#), [53](#)  
dispRity.per.group, [30](#)  
dispRity.through.time, [31](#)  
dist, [4](#)  
dist.hamming, [12](#), [13](#)  
dtt, [32](#), [33](#)  
dtt.dispRity, [32](#)

ellipse.volume (dispRity.metric), [25](#)  
extinction.subsets, [33](#)  
extract.dispRity, [34](#), [40](#), [70](#)

fetch.matrix (matrix.dispRity), [43](#)  
fill.dispRity, [35](#)  
func.div (dispRity.metric), [25](#)  
func.eve (dispRity.metric), [25](#)

gen.seq.HKY, [65](#), [66](#)  
geomorph.data.frame, [36](#)  
geomorph.ordination, [36](#)  
get.ancestors, [37](#), [53](#)  
get.bin.ages, [38](#)  
get.contrast.matrix, [13](#), [39](#)

- get.dispRity, (get.subsets), 40
- get.subsets, 35, 40, 70
- getParent, 37, 38
- glm, 75
- gpagen, 36, 37
  
- hist, 58
  
- lm, 75
  
- make.dispRity, 41
- make.metric, 23, 25, 27, 42
- MakeMorphMatrix, 16
- matrix.dispRity, 43
- mean, 26
- median, 22, 26, 58, 73
- mode.val (dispRity.metric), 25
- model.test, 44, 47, 49, 50, 52
- model.test.sim, 46, 47, 51, 52
- model.test.wrapper, 46, 49, 50
- MorphDistMatrix, 16
- morphol.disparity, 37
  
- n.ball.volume (dispRity.metric), 25
- neighbours (dispRity.metric), 25
- NJ, 12, 13
- nodes.coordinates, 26, 38, 52
- null.test, 9, 53, 58, 72, 75
  
- optim, 44, 46
- optim.parsimony, 12, 13
  
- p.adjust, 74, 75
- pair.plot, 55, 59, 75
- pairwise.dist (dispRity.metric), 25
- parsimony, 13
- pcoa, 26
- phyDat, 12, 13
- plot, 55, 58
- plot.char.diff, 12, 56
- plot.dispRity, 24, 30–33, 45, 46, 49, 51, 52, 57, 70, 74
- polygon, 58
- prcomp, 36, 37
- print.dispRity, 60
- prod, 26
  
- quantiles (dispRity.metric), 25
  
- radius (dispRity.metric), 25
  
- random.circle, 61
- randtest, 54, 58
- ranges (dispRity.metric), 25
- read.nexus.data, 16
- ReadMorphNexus, 16
- reduce.matrix, 62
- remove.zero.brlen, 63, 69
- rescale.dispRity, 64
- RI, 13
- rTraitDisc, 65, 66
  
- scale, 65
- screepplot, 54
- sim.char, 32
- sim.morpho, 6, 13, 65
- size.subsets, 67
- slice.tree, 15, 68, 77
- slide.nodes, 63, 69
- sort.dispRity, 70
- space.maker, 53, 54, 61, 71
- span.tree.length (dispRity.metric), 25
- spantree, 27
- sum, 8, 26, 42
- summary, 73
- summary.dispRity, 10, 15, 21, 24, 30–32, 45, 46, 49, 51, 52, 59, 73
  
- test.dispRity, 4, 9, 33, 34, 54, 55, 65, 70, 72, 74
- time.series (chrono.subsets), 13
- time.subsets (chrono.subsets), 13
- timeSliceTree, 68
- tree.age, 15, 18, 20, 76
  
- variances, 8, 42
- variances (dispRity.metric), 25
- vegdist, 3, 25, 26, 62