The ape Package

May 8, 2008

Version 2.2

Date 2008-05-07

Title Analyses of Phylogenetics and Evolution

Author Emmanuel Paradis, Ben Bolker, Julien Claude, Hoa Sien Cuong, Richard Desper, Benoit Durand, Julien Dutheil, Olivier Gascuel, Gangolf Jobb, Christoph Heibl, Vincent Lefort, Jim Lemon, Yvonnick Noel, Johan Nylander, Rainer Opgen-Rhein, Korbinian Strimmer, Damien de Vienne

Maintainer Emmanuel Paradis <Emmanuel.Paradis@mpl.ird.fr>

Depends R (>= 2.6.0)

Suggests gee, nlme, lattice

ZipData no

Description ape provides functions for reading, writing, plotting, and manipulating phylogenetic trees, analyses of comparative data in a phylogenetic framework, analyses of diversification and macroevolution, computing distances from allelic and nucleotide data, reading nucleotide sequences, and several tools such as Mantel's test, computation of minimum spanning tree, the population parameter theta based on various approaches, nucleotide diversity, generalized skyline plots, estimation of absolute evolutionary rates and clock-like trees using mean path lengths, non-parametric rate smoothing and penalized likelihood, classifying genes in trees using the Klastorin-Misawa-Tajima approach. Phylogeny estimation can be done with the NJ, BIONJ, ME, and ML methods.

License GPL (>= 2)

URL http://ape.mpl.ird.fr/

R topics documented:

`DNAbin` *Manipulate DNA Sequences in Bit-Level Format*

Description

These functions help to manipulate DNA sequences coded in the bit-level coding scheme.

Usage

```
## S3 method for class 'DNAbin':
print(x, ...)
## S3 method for class 'DNAbin':
summary(object, printlen = 6, digits = 3, ...)
## S3 method for class 'DNAbin':
rbind(...)
## S3 method for class 'DNAbin':
cbind(..., check.names = TRUE)
## S3 method for class 'DNAbin':
x[i, j, drop = TRUE]
## S3 method for class 'DNAbin':
as.matrix(x, ...)
```

Arguments

`x, object`	an object of class `"DNAbin"`.
`...`	either further arguments to be passed to or from other methods in the case of `print`, `summary`, and `as.matrix`, or a series of objects of class `"DNAbin"` in the case of `rbind` and `cbind`.
`printlen`	the number of labels to print (6 by default).
`digits`	the number of digits to print (3 by default).
`check.names`	a logical specifying whether to check the rownames before binding the columns (see details).
`i, j`	indices of the rows and/or columns to select or to drop. They may be numeric, logical, or character (in the same way than for standard R objects).
`drop`	logical; if `TRUE` (the default), the returned object is of the lowest possible dimension.

Details

These are all 'methods' of generic functions which are here applied to DNA sequences stored as objects of class `"DNAbin"`. They are used in the same way than the standard R functions to manipulate vectors, matrices, and lists. Additionally, the operators `[[` and `$` may be used to extract a vector from a list.

These functions are provided to manipulate easily DNA sequences coded with the bit-level coding scheme. The latter allows much faster comparisons of sequences, as well as storing them in less memory compared to the format used before **ape** 1.10.

For `cbind`, if `"check.names = TRUE"`, the rownames of each matrix are checked, and the rows are reordered if necessary. If the rownames differ among matrices, an error occurs. If `"check.names = FALSE"`, the matrices are simply binded and the rownames of the first matrix are used.

`as.matrix` may be used to convert DNA sequences (of the same length) stored in a list into a matrix while keeping the names and the class.

Value

an object of class `"DNAbin"` in the case of `rbind`, `cbind`, and `[`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. (2007) A Bit-Level Coding Scheme for Nucleotides. `http://ape.mpl.ird.fr/misc/BitLevelCodingScheme_20April2007.pdf`

See Also

`as.DNAbin`, `read.dna`, `read.GenBank`, `write.dna`

The corresponding generic functions are documented in the package **base**.

Examples

```
data(woodmouse)
woodmouse
summary(woodmouse)
summary(woodmouse, 15, 6)
summary(woodmouse[1:5, 1:300], 15, 6)
### Just to show how distances could be influenced by sampling:
dist.dna(woodmouse[1:2, ])
dist.dna(woodmouse[1:3, ])
```

`DNAmodel` *Defines Models of DNA Evolution*

Description

This function defines a model of evolution for a set of DNA sequences with possible partitions.

Usage

```
DNAmodel(model = "K80", partition = 1,
         ncat.isv = 1, invar = FALSE,
         equal.isv = TRUE, equal.invar = 1)
```

Arguments

`model`	a vector of mode character giving the substition model.
`partition`	a vector of integers defining the partitions for the substition models (eventually recycled).
`ncat.isv`	the number of categories in each partition.
`invar`	a logical value specifying whether there are invariants.
`equal.isv`	a logical value specifying whether the ‘alpha’ parameter is the same in all partitions; has no effet if `ncat = 1` or if `partition = 1`.
`equal.invar`	similar to the argument above but for the proportion of invariants.

Details

`partition` is recycled along the sequence: thus by default there is a single partition. For instance, to partition a sequence of 1000 sites into two partitions of equal length, one will use `partition = c(rep(1, 500), rep(2, 500))`. The partitions must be numbered with a series of integers (1, 2, 3, ...). To partition the codon positions, one could do `partition = c(1, 1, 2)`.

The substition models are the same in all partitions. Branch lengths are the same in all partitions up to a multiplying coefficient (the contrast parameter, denoted ’xi’).

The substitution models must be among the followings: `"JC69"` `"K80"`, `"F81"`, `"F84"`, `"HKY85"`, `"T92"`, `"TN93"`, and `"GTR"`. These models (except HKY85 and GTR) are described in the help page of `dist.dna`.

Inter-sites variation in substitution rates (ISV) is allowed by specifying `ncat.isv` greater than one.

Value

an object of class `"DNAmodel"` with components defined by the arguments of the function call.

Note

The result of this function is not intended to be used by the user, but rather to be passed to `mlphylo`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`mlphylo`, `dist.dna`

Examples

```
### the K80 model:
mod <- DNAmodel()
### the simplest substitution model:
mod <- DNAmodel("JC69")
### the classical GTR + G4 + I:
mod <- DNAmodel("GTR", ncat.isv = 4, invar = TRUE)
### codon-partitioning (with K80):
mod <- DNAmodel(partition = c(1, 1, 2))
### the same but adding inter-sites variation (the alpha parameter
### is the same for both partitions):
mod <- DNAmodel(partition = c(1, 1, 2), ncat.isv = 4)
### ... and with different `alpha' for each partition:
mod <- DNAmodel(partition = c(1, 1, 2), ncat.isv = 4, equal.isv = FALSE)
```

`GC.content` *Content in GC from DNA Sequences*

Description

This function computes the percentage of G+C in a sample of DNA sequences.

Usage

```
GC.content(x)
```

Arguments

`x`	a vector, a matrix, a data frame, or a list which contains the DNA sequences.

Details

The percentage of G+C is computed over all sequences in the sample. All missing or unknown sites are discarded from the computations. The present function actually uses the function `base.freq`.

Value

A single numeric value is returned.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`base.freq`, `seg.sites`, `nuc.div`

`Initialize.corPhyl` *Initialize a 'corPhyl' Structure Object*

Description

Initialize a `corPhyl` correlation structure object. Does the same as `Initialize.corStruct`, but also checks the row names of data and builds an index.

Usage

```
## S3 method for class 'corPhyl':
Initialize(object, data, ...)
```

Arguments

`object`	An object inheriting from class `corPhyl`.
`data`	The data to use. If it contains rownames, they are matched with the tree tip labels, otherwise data are supposed to be in the same order than tip labels and a warning is sent.
`...`	some methods for this generic require additional arguments. None are used in this method.

Value

An initialized object of same class as `object`.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

See Also

`corClasses`, `Initialize.corStruct`.

`Moran.I` *Moran's I Autocorrelation Index*

Description

This function computes Moran's I autocorrelation coefficient of `x` giving a matrix of weights using the method described by Gittleman and Kot (1990).

Usage

```
Moran.I(x, weight, scaled = FALSE, na.rm = FALSE,
        alternative = "two.sided")
```

Arguments

`x`	a numeric vector.
`weight`	a matrix of weights.
`scaled`	a logical indicating whether the coefficient should be scaled so that it varies between -1 and +1 (default to `FALSE`).
`na.rm`	a logical indicating whether missing values should be removed.
`alternative`	a character string specifying the alternative hypothesis that is tested against the null hypothesis of no phylogenetic correlation; must be of one "two.sided", "less", or "greater", or any unambiguous abbrevation of these.

Details

The matrix `weight` is used as "neighbourhood" weights, and Moran's I coefficient is computed using the formula:

$$I = \frac{n}{S_0} \frac{\sum_{i=1}^n \sum_{j=1}^n w_{i,j}(y_i - \overline{y})(y_j - \overline{y})}{\sum_{i=1}^n (y_i - \overline{y})^2}$$

with

- y_i = observations
- $w_{i,j}$ = distance weight
- n = number of observations
- S_0 = $\sum_{i=1}^n \sum_{j=1}^n wij$

The null hypothesis of no phylogenetic correlation is tested assuming normality of I under this null hypothesis. If the observed value of I is significantly greater than the expected value, then the values of `x` are positively autocorrelated, whereas if Iobserved < Iexpected, this will indicate negative autocorrelation.

Value

A list containing the elements:

`observed`	the computed Moran's I.
`expected`	the expected value of I under the null hypothesis.
`sd`	the standard deviation of I under the null hypothesis.
`p.value`	the P-value of the test of the null hypothesis against the alternative hypothesis specified in `alternative`.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩ and Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Gittleman, J. L. and Kot, M. (1990) Adaptation: statistics and a null model for estimating phylogenetic effects. *Systematic Zoology*, **39**, 227–241.

See Also

`weight.taxo`

Examples

```
tr <- rtree(30)
x <- rnorm(30)
## weights w[i,j] = 1/d[i,j]:
w <- 1/cophenetic(tr)
## set the diagonal w[i,i] = 0 (instead of Inf...):
diag(w) <- 0
Moran.I(x, w)
Moran.I(x, w, alt = "l")
Moran.I(x, w, alt = "g")
Moran.I(x, w, scaled = TRUE) # usualy the same
```

`NPRS.criterion` *Objective Function Employed in Nonparametric Rate Smoothing*

Description

`NPRS.criterion` computes the objective function to be minimized in the NPRS (nonparametric rate smoothing) algorithm described in Sanderson (1997).

Usage

```
NPRS.criterion(phy, chrono, expo = 2, minEdgeLength = 1e-06)
```

Arguments

`phy`	A non-clock-like phylogenetic tree (i.e. an object of class `"phylo"`), where the branch lengths are measured in substitutions.
`chrono`	A chronogram, i.e. a clock-like tree (i.e. an object of class `"phylo"`), where the branch lengths are measured in absolute time.
`expo`	Exponent in the objective function (default value: 2)
`minEdgeLength`	Minimum edge length in the phylogram (default value: 1e-06). If any branch lengths are smaller then they will be set to this value.

Details

Please refer to Sanderson (1997) for mathematical details. Note that is is not computationally efficient to optimize the branch lengths in a chronogram by using `NPRS.criterion` - please use `chronogram` instead.

Value

`NPRS.criterion` returns the value of the objective function given a phylogram and a chronogram.

Author(s)

Gangolf Jobb (`http://www.treefinder.de`) and Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

References

Sanderson, M. J. (1997) A nonparametric approach to estimating divergence times in the absence of rate constancy. *Molecular Biology and Evolution*, **14**, 1218–1231.

See Also

`ratogram`, `chronogram`

Examples

```
# get tree
data("landplants.newick") # example tree in NH format
tree.landplants <- read.tree(text = landplants.newick)

# plot tree
tree.landplants
plot(tree.landplants, label.offset = 0.001)

# estimate chronogram
chrono.plants <- chronogram(tree.landplants)

# plot
```

```
plot(chrono.plants, label.offset = 0.001)

# value of NPRS function for our estimated chronogram
NPRS.criterion(tree.landplants, chrono.plants)
```

---

ace *Ancestral Character Estimation*

Description

This function estimates ancestral character states, and the associated uncertainty, for continuous and discrete characters.

`logLik`, `deviance`, and `AIC` are generic functions used to extract the log-likelihood, the deviance (-2*logLik), or the Akaike information criterion of a tree. If no such values are available, `NULL` is returned.

`anova` is another generic function that is used to compare nested models: the significance of the additional parameter(s) is tested with likelihood ratio tests. You must ensure that the models are effectively nested (if they are not, the results will be meaningless). It is better to list the models from the smallest to the largest.

Usage

```
ace(x, phy, type = "continuous", method = "ML", CI = TRUE,
    model = if (type == "continuous") "BM" else "ER",
    scaled = TRUE, kappa = 1, corStruct = NULL, ip = 0.1)
## S3 method for class 'ace':
logLik(object, ...)
## S3 method for class 'ace':
deviance(object, ...)
## S3 method for class 'ace':
AIC(object, ..., k = 2)
## S3 method for class 'ace':
anova(object, ...)
```

Arguments

`x`	a vector or a factor.
`phy`	an object of class `"phylo"`.
`type`	the variable type; either `"continuous"` or `"discrete"` (or an abbreviation of these).
`method`	a character specifying the method used for estimation. Three choices are possible: `"ML"`, `"pic"`, or `"GLS"`.
`CI`	a logical specifying whether to return the 95% confidence intervals of the ancestral state estimates (for continuous characters) or the likelihood of the different states (for discrete ones).

`model`	a character specifying the model (ignored if `method = "GLS"`), or a numeric matrix if `type = "discrete"` (see details).
`scaled`	a logical specifying whether to scale the contrast estimate (used only if `method = "pic"`).
`kappa`	a positive value giving the exponent transformation of the branch lengths (see details).
`corStruct`	if `method = "GLS"`, specifies the correlation structure to be used (this also gives the assumed model).
`ip`	the initial value(s) used for the ML estimation procedure when `type == "discrete"` (possibly recycled).
`object`	an object of class `"ace"`.
`k`	a numeric value giving the penalty per estimated parameter; the default is `k = 2` which is the classical Akaike information criterion.
`...`	further arguments passed to or from other methods.

Details

If `type = "continuous"`, the default model is Brownian motion where characters evolve randomly following a random walk. This model can be fitted by maximum likelihood (the default, Schluter et al. 1997), least squares (`method = "pic"`, Felsenstein 1985), or generalized least squares (`method = "GLS"`, Martins and Hansen 1997, Cunningham et al. 1998). In the latter case, the specification of `phy` and `model` are actually ignored: it is instead given through a correlation structure with the option `corStruct`.

For discrete characters (`type = "discrete"`), only maximum likelihood estimation is available (Pagel 1994). The model is specified through a numeric matrix with integer values taken as indices of the parameters. The numbers of rows and of columns of this matrix must be equal, and are taken to give the number of states of the character. For instance, `matrix(c(0, 1, 1, 0), 2)` will represent a model with two character states and equal rates of transition, `matrix(c(0, 1, 2, 0), 2)` a model with unequal rates, `matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3)` a model with three states and equal rates of transition (the diagonal is always ignored). There are short-cuts to specify these models: `"ER"` is an equal-rates model (e.g., the first and third examples above), `"ARD"` is an all-rates-different model (the second example), and `"SYM"` is a symmetrical model (e.g., `matrix(c(0, 1, 2, 1, 0, 3, 2, 3, 0), 3)`). If a short-cut is used, the number of states is determined from the data.

Value

a list with the following elements:

`ace`	if `type = "continuous"`, the estimates of the ancestral character values.
`CI95`	if `type = "continuous"`, the estimated 95% confidence intervals.
`sigma2`	if `type = "continuous"`, `model = "BM"`, and `method = "ML"`, the maximum likelihood estimate of the Brownian parameter.
`rates`	if `type = "discrete"`, the maximum likelihood estimates of the transition rates.
`se`	if `type = "discrete"`, the standard-errors of estimated rates.

`index.matrix`	if `type = "discrete"`, gives the indices of the `rates` in the rate matrix.
`loglik`	if `method = "ML"`, the maximum log-likelihood.
`lik.anc`	if `type = "discrete"`, the scaled likelihoods of each ancestral state.
`call`	the function call.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩, Ben Bolker ⟨bolker@zoo.ufl.edu⟩

References

Cunningham, C. W., Omland, K. E. and Oakley, T. H. (1998) Reconstructing ancestral character states: a critical reappraisal. *Trends in Ecology & Evolution*, **13**, 361–366.

Felsenstein, J. (1985) Phylogenies and the comparative method. *American Naturalist*, **125**, 1–15.

Martins, E. P. and Hansen, T. F. (1997) Phylogenies and the comparative method: a general approach to incorporating phylogenetic information into the analysis of interspecific data. *American Naturalist*, **149**, 646–667.

Pagel, M. (1994) Detecting correlated evolution on phylogenies: a general method for the comparative analysis of discrete characters. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **255**, 37–45.

Schluter, D., Price, T., Mooers, A. O. and Ludwig, D. (1997) Likelihood of ancestor states in adaptive radiation. *Evolution*, **51**, 1699–1711.

See Also

`corBrownian`, `corGrafen`, `corMartins`, `compar.ou`, `anova`

Examples

```
### Just some random data...
data(bird.orders)
x <- rnorm(23)
### Compare the three methods for continuous characters:
ace(x, bird.orders)
ace(x, bird.orders, method = "pic")
ace(x, bird.orders, method = "GLS",
    corStruct = corBrownian(1, bird.orders))
### For discrete characters:
x <- factor(c(rep(0, 5), rep(1, 18)))
ans <- ace(x, bird.orders, type = "d")
#### Showing the likelihoods on each node:
plot(bird.orders, type = "c", FALSE, label.offset = 1)
co <- c("blue", "yellow")
tiplabels(pch = 22, bg = co[as.numeric(x)], cex = 2, adj = 1)
nodelabels(thermo = ans$lik.anc, piecol = co, cex = 0.75)
### An example of the use of the argument `ip':
tr <- character(4)
tr[1] <- "((((t10:5.03,t2:5.03):2.74,(t9:4.17,"
tr[2] <- "t5:4.17):3.60):2.80,(t3:4.05,t7:"
tr[3] <- "4.05):6.53):2.32,((t6:4.38,t1:4.38):"
tr[4] <- "2.18,(t8:2.17,t4:2.17):4.39):6.33);"
tr <- read.tree(text = paste(tr, collapse = ""))
y <- c(rep(1, 6), rep(2, 4))
### The default `ip = 0.1' makes ace fails:
ace(y, tr, type = "d")
ace(y, tr, type = "d", ip = 0.01)
### Surprisingly, using an initial value farther to the
### MLE than the default one works:
ace(y, tr, type = "d", ip = 0.3)
```

`add.scale.bar` *Add a Scale Bar to a Phylogeny Plot*

Description

This function adds a horizontal bar giving the scale of the branch lengths to a plot of a phylogenetic tree on the current graphical device.

Usage

```
add.scale.bar(x = 0, y = 1, length = NULL, ...)
```

Arguments

`x`	x location of the bar.
`y`	y location of the bar.
`length`	a numeric value giving the length of the scale bar. If none is supplied, a value is calculated from the data.
`...`	further arguments to be passed to `text`.

Details

As from version 1.4 of ape, the options of this function have been redefined, and have now default values. By default, the scale bar is plotted on the left bottom corner of the plot.

The further arguments (`...`) are used to format the text. They may be `font`, `cex`, `col`, and so on (see examples below, and the help page on `text`).

The function `locator` may be used to determine the `x` and `y` arguments.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.phylo`, `axisPhylo`, `locator`

Examples

```
tr <- rtree(10)
layout(matrix(1:2, 2, 1))
plot(tr)
add.scale.bar()
plot(tr)
add.scale.bar(cex = 0.7, font = 2, col = "red")
layout(matrix(1))
```

---

all.equal.phylo   *Global Comparison of two Phylogenies*

Description

This function makes a global comparison of two phylogenetic trees.

Usage

```
## S3 method for class 'phylo':
all.equal(target, current, use.edge.length = TRUE,
                          use.tip.label = TRUE, index.return = FALSE,
                          tolerance = .Machine$double.eps ^ 0.5,
                          scale = NULL, ...)
```

Arguments

`target`	an object of class "`phylo`".
`current`	an object of class "`phylo`".
`use.edge.length`	if `FALSE` only the topologies are compared; the default is `TRUE`.
`use.tip.label`	if `FALSE` the unlabelled trees are compared; the default is `TRUE`.
`index.return`	if `TRUE` the function returns a two-column matrix giving the correspondence between the nodes of both trees.
`tolerance`	the numeric tolerance used to compare the branch lengths.
`scale`	a positive number, comparison of branch lengths is made after scaling (i.e., dividing) them by this number.
`...`	further arguments passed to or from other methods.

Details

This function is meant to be an adaptation of the generic function `all.equal` for the comparison of phylogenetic trees.

A single phylogenetic tree may have several representations in the Newick format and in the "`phylo`" class of objects used in 'ape'. One aim of the present function is to be able to identify whether two objects of class "`phylo`" represent the same phylogeny.

Only the labelled topologies are compared (i.e. branch lengths are not considered.

Value

A logical value, or a two-column matrix.

Author(s)

Benoît ⟨b.durand@alfort.AFSSA.FR⟩

See Also

`all.equal` for the generic R function

Examples

```
### maybe the simplest example of two representations
### for the same rooted tree...:
t1 <- read.tree(text = "(a:1,b:1);")
t2 <- read.tree(text = "(b:1,a:1);")
all.equal(t1, t2)
### ... compare with this:
identical(t1, t2)
### one just slightly more complicated...:
t3 <- read.tree(text = "((a:1,b:1):1,c:2);")
t4 <- read.tree(text = "(c:2,(a:1,b:1):1);")
all.equal(t3, t4) # == all.equal.phylo(t3, t4)
### ... here we force the comparison as lists:
all.equal.list(t3, t4)
t5 <- read.tree(text = "(a:2,(c:1,b:1):1);")
### note that this does NOT return FALSE...:
all.equal(t3, t5)
### ... do this instead:
identical(all.equal(t3, t5), TRUE)
```

`ape-internal` *Internal Ape Functions*

Description

Internal ape functions.

Note

These are not to be called by the user (or in some cases are just waiting for proper documentation to be written).

as.alignment — *Conversion Among DNA Sequence Internal Formats*

Description

These functions transform a set of DNA sequences among various internal formats.

Usage

```
as.alignment(x)
as.DNAbin(x, ...)

## S3 method for class 'character':
as.DNAbin(x, ...)

## S3 method for class 'list':
as.DNAbin(x, ...)

## S3 method for class 'alignment':
as.DNAbin(x, ...)

## S3 method for class 'DNAbin':
as.character(x, ...)
```

Arguments

`x`	a matrix or a list containing the DNA sequences, or an object of class `"alignment"`.
`...`	further arguments to be passed to or from other methods.

Details

For `as.alignment`, the sequences given as argument should be stored as matrices or lists of single-character strings (the format used in **ape** before version 1.10). The returned object is in the format used in the package **seqinr** to store aligned sequences.

`as.DNAbin` is a generic function with methods so that it works with sequences stored into vectors, matrices, or lists.

`as.character` is a generic function: the present method converts objects of class `"DNAbin"` into the format used before **ape** 1.10 (matrix of single characters, or list of vectors of single characters). This function must be used first to convert objects of class `"DNAbin"` into the class `"alignment"`.

Value

an object of class `"alignment"` in the case of `"as.alignment"`; an object of class `"DNAbin"` in the case of `"as.DNAbin"`; a matrix of mode character or a list containing vectors of mode character in the case of `"as.character"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`DNAbin`, `read.dna`, `read.GenBank`, `write.dna`

Examples

```
data(woodmouse)
x <- as.character(woodmouse)
x[, 1:20]
str(as.alignment(x))
identical(as.DNAbin(x), woodmouse)
```

`as.matching` *Conversion Between Phylo and Matching Objects*

Description

These functions convert objects between the classes `"phylo"` and `"matching"`.

Usage

```
as.matching(x, ...)
## S3 method for class 'phylo':
as.matching(x, labels = TRUE, ...)
## S3 method for class 'matching':
as.phylo(x, ...)
```

Arguments

`x`	an object to convert as an object of class `"matching"` or of class `"phylo"`.
`labels`	a logical specifying whether the tip and node labels should be included in the returned matching.
`...`	further arguments to be passed to or from other methods.

Details

A matching is a representation where each tip and each node are given a number, and sibling groups are grouped in a “matching pair” (see Diaconis and Holmes 1998, for details). This coding system can be used only for binary (fully dichotomous) trees.

Diaconis and Holmes (1998) gave some conventions to insure that a given tree has a unique representation as a matching. I have tried to follow them in the present functions.

Value

`as.matching` returns an object of class `"matching"` with the following component:

`matching`	a three-column numeric matrix where the first two columns represent the sibling pairs, and the third one the corresponding ancestor.
`tip.label`	(optional) a character vector giving the tip labels where the ith element is the label of the tip numbered i in `matching`.
`node.label`	(optional) a character vector giving the node labels in the same order than in `matching` (i.e. the ith element is the label of the node numbered i + n in `matching`, with n the number of tips).

`as.phylo.matching` returns an object of class `"phylo"`.

Note

Branch lengths are not supported in the present version.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Diaconis, P. W. and Holmes, S. P. (1998) Matchings and phylogenetic trees. *Proceedings of the National Academy of Sciences USA*, **95**, 14600–14602.

See Also

`as.phylo`

Examples

```
data(bird.orders)
m <- as.matching(bird.orders)
str(m)
m
tr <- as.phylo(m)
all.equal(tr, bird.orders, use.edge.length = FALSE)
```

as.phylo *Conversion Among Tree Objects*

Description

`as.phylo` is a generic function which converts an object into a tree of class `"phylo"`. There are currently two methods for this generic for objects of class `"hclust"` and of class `"phylog"` (implemented in the package ade4). `as.hclust.phylo` is a method of the generic `as.hclust` which converts an object of class `"phylo"` into one of class `"hclust"`. This can used to convert an object of class `"phylo"` into one of class `"dendrogram"` (see examples).

`old2new.phylo` and `new2old.phylo` are utility functions for converting between the old and new coding of the class `"phylo"`.

Usage

```
as.phylo(x, ...)
## S3 method for class 'hclust':
as.phylo(x, ...)
## S3 method for class 'phylog':
as.phylo(x, ...)
## S3 method for class 'phylo':
as.hclust(x, ...)
old2new.phylo(phy)
new2old.phylo(phy)
```

Arguments

`x`	an object to be converted into another class.
`...`	further arguments to be passed to or from other methods.
`phy`	an object of class `"phylo"`.

Value

An object of class `"hclust"` or `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`hclust`, `as.hclust`, `dendrogram`, `phylog`, `as.phylo.formula`

Examples

```
data(bird.orders)
hc <- as.hclust(bird.orders)
tr <- as.phylo(hc)
identical(bird.orders, tr) # FALSE, but...
all.equal(bird.orders, tr) # ... TRUE

### shows the three plots for tree objects:
dend <- as.dendrogram(hc)
layout(matrix(c(1:3, 3), 2, 2))
plot(bird.orders, font = 1)
plot(hc)
par(mar = c(8, 0, 0, 0)) # leave space for the labels
plot(dend)

### how to get (nearly) identical plots with
### plot.phylo and plot.dendrogram:
layout(matrix(1:2, 2, 1))
plot(bird.orders, font = 1, no.margin = TRUE)
par(mar = c(0, 0, 0, 8))
plot((dend), horiz = TRUE)
layout(matrix(1, 1, 1))
```

`as.phylo.formula` *Conversion from Taxonomy Variables to Phylogenetic Trees*

Description

The function `as.phylo.formula` (short form `as.phylo`) builds a phylogenetic tree (an object of class `phylo`) from a set of nested taxonomic variables.

Usage

```
## S3 method for class 'formula':
as.phylo(x, data = parent.frame(), ...)
```

Arguments

`x`	a right-side formula describing the taxonomic relationship: `~C1/C2/.../Cn`.
`data`	the data.frame where to look for the variables (default to environment).
`...`	further arguments to be passed from other methods.

Details

Taxonomic variables must be nested and passed in the correct order: the higher clade must be on the left of the formula, for instance `~Order/Family/Genus/Species`. In most cases, the resulting tree will be unresolved and contain polytomies.

Value

An object of class `phylo`.

Author(s)

Julien Dutheil ⟨Julien.Dutheil@univ-montp2.fr⟩

See Also

`as.phylo`, `read.tree` for a description of `phylo` objects, `multi2di`

Examples

```
data(carnivora)
plot(as.phylo(~SuperFamily/Family/Genus/Species, data=carnivora))
```

`axisPhylo` *Axis on Side of Phylogeny*

Description

This function adds a scaled axis on the side of a phylogeny plot.

Usage

```
axisPhylo(side = 1, ...)
```

Arguments

`side`	a numeric value specifying the side where the axis is plotted: 1: below, 2: left, 3: above, 4: right.
`...`	further arguments to be passed to `axis`.

Details

The further arguments (`...`) are used to format the axis. They may be `font`, `cex`, `col`, `las`, and so on (see the help pages on `axis` and `par`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.phylo`, `add.scale.bar`, `axis`, `par`

Examples

```
tr <- rtree(30)
ch <- rcoal(30)
plot(ch)
axisPhylo()
plot(tr, "c", FALSE, direction = "u")
axisPhylo(2, las = 1)
```

`balance` *Balance of a Dichotomous Phylogenetic Tree*

Description

This function computes the balance of a phylogenetic tree, that is for each node of the tree the numbers of descendants (i.e. tips) on each of its daughter-branch. The tree must be fully dichotomous.

Usage

```
balance(phy)
```

Arguments

`phy`	an object of class `"phylo"`.

Value

a numeric matrix with two columns and one row for each node of the tree. The columns give the numbers of descendants on each daughter-branches (the order of both columns being arbitrary). If the phylogeny `phy` has an element `node.label`, this is used as rownames for the returned matrix; otherwise the numbers (of mode character) of the matrix `edge` of `phy` are used as rownames.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Aldous, D. J. (2001) Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. *Statistical Science*, **16**, 23–34.

`base.freq` *Base frequencies from DNA Sequences*

Description

This function computes the relative frequencies (i.e. percentages) of the four DNA bases (adenine, cytosine, guanine, and thymidine) from a sample of sequences.

Usage

```
base.freq(x)
```

Arguments

`x`	a vector, a matrix, or a list which contains the DNA sequences.

Details

The base frequencies are computed over all sequences in the sample. All missing or unknown sites are discarded from the computations.

Value

A numeric vector stoting the relative frequencies with names `c("a", "c", "g", "t")`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`GC.content`, `seg.sites`, `nuc.div`, `DNAbin`

`bd.ext` *Extended Version of the Birth-Death Models to Estimate Speciation and Extinction Rates*

Description

This function fits by maximum likelihood a birth-death model to the combined phylogenetic and taxonomic data of a given clade. The phylogenetic data are given by a tree, and the taxonomic data by the number of species for the its tips.

Usage

```
bd.ext(phy, S)
```

Arguments

`phy`	an object of class `"phylo"`.
`S`	a numeric vector giving the number of species for each tip.

Details

A re-parametrization of the birth-death model studied by Kendall (1948) so that the likelihood has to be maximized over *d/b* and *b* - *d*, where *b* is the birth rate, and *d* the death rate.

The standard-errors of the estimated parameters are computed using a normal approximation of the maximum likelihood estimates.

If the argument `S` has names, then they are matched to the tip labels of `phy`. The user must be careful here since the function requires that both series of names perfectly match, so this operation may fail if there is a typing or syntax error. If both series of names do not match, the values `S` are taken to be in the same order than the tip labels of `phy`, and a warning message is issued.

Note that the function does not check that the tree is effectively ultrametric, so if it is not, the returned result may not be meaningful.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. (2003) Analysis of diversification: combining phylogenetic and taxonomic data. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **270**, 2499–2505.

See Also

`birthdeath`, `branching.times`, `diversi.gof`, `diversi.time`, `ltt.plot`, `yule`, `yule.cov`

Examples

```
### An example from Paradis (2003) using the avian orders:
data(bird.orders)
### Number of species in each order from Sibley and Monroe (1990):
S <- c(10, 47, 69, 214, 161, 17, 355, 51, 56, 10, 39, 152,
       6, 143, 358, 103, 319, 23, 291, 313, 196, 1027, 5712)
bd.ext(bird.orders, S)
```

`bind.tree` *Binds Trees*

Description

This function binds together two phylogenetic trees to give a single object of class `"phylo"`.

Usage

```
bind.tree(x, y, where = "root", position = 0)
```

Arguments

`x`	an object of class `"phylo"`.
`y`	an object of class `"phylo"`.
`where`	an) integer giving the number of the node or tip of the tree `x` where the tree `y` is binded (`"root"` is a short-cut for the root).
`position`	a numeric value giving the position from the tip or node given by `node` where the tree `y` is binded; negative values are ignored.

Details

The argument `x` can be seen as the receptor tree, whereas `y` is the donor tree. The root of `y` is then sticked on a location of `x` specified by `where` and, possibly, `position`. If `y` has a root edge, this is added as in internal branch in the resulting tree.

Value

an object of class `"phylo"`.

Note

For the moment, this function handles only trees with branch lengths, and does not handle node labels.

Further testing/improvements may be needed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`drop.tip`, `root`

Examples

```
### binds the two clades of bird orders
cat("((Struthioniformes:21.8,Tinamiformes:21.8):4.1,",
    "((Craciformes:21.6,Galliformes:21.6):1.3,Anseriformes:22.9):3.0):2.1;",
    file = "ex1.tre", sep = "\n")
cat("(Turniciformes:27.0,(Piciformes:26.3,((Galbuliformes:24.4,",
    "((Bucerotiformes:20.8,Upupiformes:20.8):2.6,",
    "(Trogoniformes:22.1,Coraciiformes:22.1):1.3):1.0):0.6,",
    "(Coliiformes:24.5,(Cuculiformes:23.7,(Psittaciformes:23.1,",
    "(((Apodiformes:21.3,Trochiliformes:21.3):0.6,",
    "(Musophagiformes:20.4,Strigiformes:20.4):1.5):0.6,",
    "((Columbiformes:20.8,(Gruiformes:20.1,Ciconiiformes:20.1):0.7):0.8,",
    "Passeriformes:21.6):0.9):0.6):0.6):0.8):0.5):1.3):0.7):1.0;",
    file = "ex2.tre", sep = "\n")
tree.bird1 <- read.tree("ex1.tre")
tree.bird2 <- read.tree("ex2.tre")
unlink(c("ex1.tre", "ex2.tre")) # clean-up
birds <- bind.tree(tree.bird1, tree.bird2, where = "root",
                   position = tree.bird1$root.edge)
birds
layout(matrix(c(1, 2, 3, 3), 2, 2))
plot(tree.bird1)
plot(tree.bird2)
plot(birds)
layout(matrix(1))
```

`BIONJ` *Tree Estimation Based on an Improved Version of the NJ Algorithm*

Description

This function performs the BIONJ algorithm of Gascuel (1997).

Usage

```
bionj(X)
```

Arguments

`X`	a distance matrix; may be an object of class `"dist"`.

Value

an object of class `"phylo"`.

Author(s)

original C code by Hoa Sien Cuong and Olivier Gascuel; adapted and ported to R by Vincent Lefort ⟨vincent.lefort@lirmm.fr⟩

References

Gascuel, O. (1997) BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, **14:**, 685–695.

See Also

nj, fastme, write.tree, read.tree, dist.dna, mlphylo

Examples

```
### From Saitou and Nei (1987, Table 1):
x <- c(7, 8, 11, 13, 16, 13, 17, 5, 8, 10, 13,
       10, 14, 5, 7, 10, 7, 11, 8, 11, 8, 12,
       5, 6, 10, 9, 13, 8)
M <- matrix(0, 8, 8)
M[row(M) > col(M)] <- x
M[row(M) < col(M)] <- x
rownames(M) <- colnames(M) <- 1:8
tr <- bionj(M)
plot(tr, "u")
### a less theoretical example
data(woodmouse)
trw <- bionj(dist.dna(woodmouse))
plot(trw)
```

---

bird.families *Phylogeny of the Families of Birds From Sibley and Ahlquist*

Description

This data set describes the phylogenetic relationships of the families of birds as reported by Sibley and Ahlquist (1990). Sibley and Ahlquist inferred this phylogeny from an extensive number of DNA/DNA hybridization experiments. The “tapestry” reported by these two authors (more than 1000 species out of the ca. 9000 extant bird species) generated a lot of debates.

The present tree is based on the relationships among families. A few families were not included in the figures in Sibley and Ahlquist, and thus are not included here as well. The branch lengths were calculated from the values of $\Delta T_{50}H$ as found in Sibley and Ahlquist (1990, figs. 354, 355, 356, and 369).

Usage

```
data(bird.families)
```

Format

The data are stored as an object of class `"phylo"` which structure is described in the help page of the function read.tree.

Source

Sibley, C. G. and Ahlquist, J. E. (1990) Phylogeny and classification of birds: a study in molecular evolution. New Haven: Yale University Press.

See Also

`read.tree`, `bird.orders`

Examples

```
data(bird.families)
op <- par()
par(cex = 0.3)
plot(bird.families)
par(op)
```

`bird.orders` *Phylogeny of the Orders of Birds From Sibley and Ahlquist*

Description

This data set describes the phylogenetic relationships of the orders of birds as reported by Sibley and Ahlquist (1990). Sibley and Ahlquist inferred this phylogeny from an extensive number of DNA/DNA hybridization experiments. The “tapestry” reported by these two authors (more than 1000 species out of the ca. 9000 extant bird species) generated a lot of debates.

The present tree is based on the relationships among orders. The branch lengths were calculated from the values of $\Delta T_{50}H$ as found in Sibley and Ahlquist (1990, fig. 353).

Usage

```
data(bird.orders)
```

Format

The data are stored as an object of class `"phylo"` which structure is described in the help page of the function `read.tree`.

Source

Sibley, C. G. and Ahlquist, J. E. (1990) Phylogeny and classification of birds: a study in molecular evolution. New Haven: Yale University Press.

See Also

`read.tree`, `bird.families`

Examples

```
data(bird.orders)
plot(bird.orders)
```

---

birthdeath — *Estimation of Speciation and Extinction Rates With Birth-Death Models*

Description

This function fits by maximum likelihood a birth-death model to the branching times computed from a phylogenetic tree using the method of Nee et al. (1994).

Usage

```
birthdeath(phy)
## S3 method for class 'birthdeath':
print(x, ...)
```

Arguments

`phy`	an object of class `"phylo"`.
`x`	an object of class `"birthdeath"`.
`...`	further arguments passed to the `print` function.

Details

Nee et al. (1994) used a re-parametrization of the birth-death model studied by Kendall (1948) so that the likelihood has to be maximized over *d/b* and *b - d*, where *b* is the birth rate, and *d* the death rate. This is the approach used by the present function.

This function computes the standard-errors of the estimated parameters using a normal approximations of the maximum likelihood estimates: this is likely to be inaccurate because of asymmetries of the likelihood function (Nee et al. 1995). In addition, 95 intervals of both parameters are computed using profile likelihood: they are particularly useful if the estimate of *d/b* is at the boundary of the parameter space (i.e. 0, which is often the case).

Note that the function does not check that the tree is effectively ultrametric, so if it is not, the returned result may not be meaningful.

Value

An object of class `"birthdeath"` which is a list with the following components:

`tree`	the name of the tree analysed.
`N`	the number of species.
`dev`	the deviance (= -2 log lik) at its minimum.
`para`	the estimated parameters.

`se`	the corresponding standard-errors.
`CI`	the 95% profile-likelihood confidence intervals.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Kendall, D. G. (1948) On the generalized “birth-and-death” process. *Annals of Mathematical Statistics*, **19**, 1–15.

Nee, S., May, R. M. and Harvey, P. H. (1994) The reconstructed evolutionary process. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, **344**, 305–311.

Nee, S., Holmes, E. C., May, R. M. and Harvey, P. H. (1995) Estimating extinctions from molecular phylogenies. in *Extinction Rates*, eds. Lawton, J. H. and May, R. M., pp. 164–182, Oxford University Press.

See Also

`branching.times`, `diversi.gof`, `diversi.time`, `ltt.plot`, `yule`, `bd.ext`, `yule.cov`

`boot.phylo` *Tree Bipartition and Bootstrapping Phylogenies*

Description

These functions analyse bipartitions found in a series of trees.

`prop.part` counts the number of bipartitions found in a series of trees given as `...`.

`prop.clades` counts the number of times the bipartitions present in `phy` are present in a series of trees given as `...` or in the list previously computed and given with `part`.

`boot.phylo` performs a bootstrap analysis.

Usage

```
boot.phylo(phy, x, FUN, B = 100, block = 1, trees = FALSE)
prop.part(..., check.labels = FALSE)
prop.clades(phy, ..., part = NULL)
## S3 method for class 'prop.part':
print(x, ...)
## S3 method for class 'prop.part':
summary(object, ...)
## S3 method for class 'prop.part':
plot(x, barcol = "blue", leftmar = 4, ...)
```

Arguments

`phy`	an object of class `"phylo"`.
`x`	in the case of `boot.phylo`: a taxa (rows) by characters (columns) matrix; this may be presented as a list; in the case of `print` and `plot`: an object of class `"prop.part"`.
`FUN`	the function used to estimate `phy` (see details).
`B`	the number of bootstrap replicates.
`block`	the number of columns in `x` that will be resampled together (see details).
`trees`	a logical specifying whether to return the bootstraped trees (`FALSE` by default).
`...`	either (i) a single object of class `"phylo"`, (ii) a series of such objects separated by commas, or (iii) a list containing such objects. In the case of `plot` further arguments for the plot (see details).
`check.labels`	a logical specifying whether to check the labels of each tree. If `FALSE` (the default), it is assumed that all trees have the same tip labels, and that they are in the same order (see details).
`part`	a list of partitions as returned by `prop.part`; if this is used then `...` is ignored.
`object`	an object of class `"prop.part"`.
`barcol`	the colour used for the bars displaying the number of partitions in the upper panel.
`leftmar`	the size of the margin on the left to display the tip labels.

Details

The argument `FUN` in `boot.phylo` must be the function used to estimate the tree from the original data matrix. Thus, if the tree was estimated with neighbor-joining (see `nj`), one maybe wants something like `FUN = function(xx) nj(dist.dna(xx))`.

`block` in `boot.phylo` specifies the number of columns to be resampled altogether. For instance, if one wants to resample at the codon-level, then `block = 3` must be used.

Using (the default) `check.labels = FALSE` in `prop.part` results in considerable decrease in computing times. This requires that (i) all trees have the same tip labels, *and* (ii) these labels are ordered similarly in all trees (in other words, the element `tip.label` are identical in all trees).

The plot function represents a contingency table of the different partitions (on the *x*-axis) in the lower panel, and their observed numbers in the upper panel. Any further arguments (...) are used to change the aspects of the points in the lower panel: these may be `pch`, `col`, `bg`, `cex`, etc. This function works only if there is an attribute `labels` in the object.

The print method displays the partitions and their numbers. The summary method extracts the numbers only.

Value

`prop.part` returns an object of class `"prop.part"` which is a list with an attribute `"number"`. The elements of this list are the observed clades, and the attribute their respective numbers. If the default `check.labels = FALSE` is used, an attribute `"labels"` is added, and the vectors of the returned object contains the indices of these labels instead of the labels themselves.

`prop.clades` and `boot.phylo` return a numeric vector which *i*th element is the number associated to the *i*th node of `phy`. If `trees = TRUE`, `boot.phylo` returns a list whose first element (named `"BP"`) is like before, and the second element (`"trees"`) is a list with the bootstraped trees.

`summary` returns a numeric vector.

Note

`prop.clades` calls internally `prop.part` with the option `check.labels = TRUE`, which may be very slow. If the trees passed as `...` fulfills conditions (i) and (ii) above, then it might be faster to first call, e.g., `pp <- prop.part(...)`, then use the option `part`: `prop.clades(phy, part = pp)`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Efron, B., Halloran, E. and Holmes, S. (1996) Bootstrap confidence levels for phylogenetic trees. *Proceedings of the National Academy of Sciences USA*, **93**, 13429–13434.

Felsenstein, J. (1985) Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, **39**, 783–791.

See Also

`dist.topo`, `consensus`, `nodelabels`

Examples

```
data(woodmouse)
tr <- nj(dist.dna(woodmouse))
### Are bootstrap values stable?
for (i in 1:5)
  print(boot.phylo(tr, woodmouse, function(xx) nj(dist.dna(xx))))
### How many partitions in 100 random trees of 10 labels?...
TR <- replicate(100, rtree(10), FALSE)
pp10 <- prop.part(TR)
length(pp10)
### ... and in 100 random trees of 20 labels?
TR <- replicate(100, rtree(20), FALSE)
pp20 <- prop.part(TR)
length(pp20)
plot(pp10, pch = "x", col = 2)
plot(pp20, pch = "x", col = 2)
```

`branching.times` *Branching Times of a Phylogenetic Tree*

Description

This function computes the branching times of a phylogenetic tree, that is the distance from each node to the tips, under the assumption that the tree is ultrametric. Note that the function does not check that the tree is effectively ultrametric, so if it is not, the returned result may not be meaningful.

Usage

```
branching.times(phy)
```

Arguments

`phy`	an object of class `"phylo"`.

Value

a numeric vector with the branching times. If the phylogeny `phy` has an element `node.label`, this is used as names for the returned vector; otherwise the numbers (of mode character) of the matrix `edge` of `phy` are used as names.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`is.ultrametric`

`carnivora` *Carnivora body sizes and life history traits*

Description

Dataset adapted from Gittleman (1986), including 2 morphological variables (body and brain sizes), 8 life history traits variables and 4 taxonomic variables.

Usage

```
data(carnivora)
```

Format

A data frame with 112 observations on 17 variables.

[,1]	Order	factor	Carnivora order
[,2]	SuperFamily	factor	Super family (Caniformia or Feliformia)
[,3]	Family	factor	Carnivora family
[,4]	Genus	factor	Carnivora genus
[,5]	Species	factor	Carnivora species
[,6]	FW	numeric	Female body weight (kg)
[,7]	SW	numeric	Average body weight of adult male and adult female (kg)
[,8]	FB	numeric	Female brain weight (g)
[,9]	SB	numeric	Average brain weight of adult male and adult female (g)
[,10]	LS	numeric	Litter size
[,11]	GL	numeric	Gestation length (days)
[,12]	BW	numeric	Birth weigth (g)
[,13]	WA	numeric	Weaning age (days)
[,14]	AI	numeric	Age of independance (days)
[,15]	LY	numeric	Longevity (months)
[,16]	AM	numeric	Age of sexual maturity (days)
[,17]	IB	numeric	Inter-birth interval (months)

Source

Gittleman, J. L. (1986) Carnivore life history patterns: allometric, phylogenetic and ecological associations. *American Naturalist*, **127**: 744–771.

Examples

```
data(carnivora);
# This is figure 1 of Gittleman 1986:
library(lattice)
trellis.device(color=FALSE)
xyplot(BW ~ FW, groups=Family, data=carnivora, auto.key=TRUE, xlog=TRUE,
    scale=list(log=TRUE), ylim=c(1, 2000))
trellis.device(color=FALSE)
xyplot(BW ~ FB, groups=Family, data=carnivora, auto.key=TRUE, xlog=TRUE,
    scale=list(log=TRUE), ylim=c(1, 2000))
```

`cherry` *Number of Cherries and Null Models of Trees*

Description

This function calculates the number of cherries (see definition below) on a phylogenetic tree, and tests the null hypotheses whether this number agrees with those predicted from two null models of trees (the Yule model, and the uniform model).

Usage

```
cherry(phy)
```

Arguments

`phy`	an object of class `"phylo"`.

Details

A cherry is a pair of adjacent tips on a tree. The tree can be either rooted or unrooted, but the present function considers only rooted trees. The probability distribution function of the number of cherries on a tree depends on the speciation/extinction model that generated the tree.

McKenzie and Steel (2000) derived the probability distribution function of the number of cherries for two models: the Yule model and the uniform model. Broadly, in the Yule model, each extant species is equally likely to split into two daughter-species; in the uniform model, a branch is added to tree on any of the already existing branches with a uniform probability.

The probabilities are computed using recursive formulae; however, for both models, the probability density function converges to a normal law with increasing number of tips in the tree. The function uses these normal approximations for a number of tips greater than or equal to 20.

Value

A NULL value is returned, the results are simply printed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

McKenzie, A. and Steel, M. (2000) Distributions of cherries for two models of trees. *Mathematical Biosciences*, **164**, 81–92.

See Also

`gammaStat`

`chiroptera` *Bat Phylogeny*

Description

This phylogeny of bats (Mammalia: Chiroptera) is a supertree (i.e. a composite phylogeny constructed from several sources; see source for details).

Usage

```
data(chiroptera)
```

Format

The data are stored in RData (binary) format.

Source

Jones, K. E., Purvis, A., MacLarnon, A., Bininda-Emonds, O. R. P. and Simmons, N. B. (2002) A phylogenetic supertree of the bats (Mammalia: Chiroptera). *Biological Reviews of the Cambridge Philosophical Society*, **77**, 223–259.

See Also

`read.nexus`, `zoom`

Examples

```
data(chiroptera)
str(chiroptera)
op <- par()
par(cex = 0.3)
plot(chiroptera, type = "c")
par(op)
```

---

chronoMPL *Molecular Dating With Mean Path Lengths*

Description

This function estimates the node ages of a tree using the mean path lengths method of Britton et al. (2002). The branch lengths of the input tree are interpreted as (mean) numbers of substitutions.

Usage

```
chronoMPL(phy, se = TRUE, test = TRUE)
```

Arguments

`phy`	an object of class `"phylo"`.
`se`	a logical specifying whether to compute the standard-errors of the node ages (`TRUE` by default).
`test`	a logical specifying whether to test the molecular clock at each node (`TRUE` by default).

Details

The mean path lengths (MPL) method estimates the age of a node with the mean of the distances from this node to all tips descending from it. Under the assumption of a molecular clock, standard-errors of the estimates node ages can be computed (Britton et al. 2002).

The tests performed if `test = TRUE` is a comparison of the MPL of the two subtrees originating from a node; the null hypothesis is that the rate of substitution was the same in both subtrees (Britton et al. 2002). The test statistic follows, under the null hypothesis, a standard normal distribution. The returned *P*-value is the probability of observing a greater absolute value (i.e., a two-sided test). No correction for multiple testing is applied: this is left to the user.

Absolute dating can be done by multiplying the edge lengths found by calibrating one node age.

Value

an object of class `"phylo"` with branch lengths as estimated by the function. There are, by default, two attributes:

`stderr`	the standard-errors of the node ages.
`Pval`	the *P*-value of the test of the molecular clock for each node.

Note

The present version requires a dichotomous tree.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Britton, T., Oxelman, B., Vinnersten, A. and Bremer, K. (2002) Phylogenetic dating with confidence intervals using mean path lengths. *Molecular Phylogenetics and Evolution*, **24**, 58–65.

See Also

`chronogram`, `ratogram`, `NPRS.criterion`, `chronopl`

Examples

```
tr <- rtree(10)
tr$edge.length <- 5*tr$edge.length
chr <- chronoMPL(tr)
layout(matrix(1:4, 2, 2, byrow = TRUE))
plot(tr)
title("The original tree")
plot(chr)
axisPhylo()
title("The dated MPL tree")
plot(chr)
nodelabels(round(attr(chr, "stderr"), 3))
```

```
title("The standard-errors")
plot(tr)
nodelabels(round(attr(chr, "Pval"), 3))
title("The tests")
```

---

chronogram *Chronogram Computed by Nonparametric Rate Smoothing*

Description

`chronogram` computes a chronogram from a phylogram by applying the NPRS (nonparametric rate smoothing) algorithm described in Sanderson (1997).

Usage

```
chronogram(phy, scale = 1, expo = 2, minEdgeLength = 1e-06)
```

Arguments

`phy`	A phylogenetic tree (i.e. an object of class `"phylo"`), where the branch lengths are measured in substitutions.
`scale`	Age of the root in the inferred chronogram (default value: 0).
`expo`	Exponent in the objective function (default value: 2)
`minEdgeLength`	Minimum edge length in the phylogram (default value: 1e-06). If any branch lengths are smaller then they will be set to this value.

Details

Please refer to Sanderson (1997) for mathematical details

Value

`chronogram` returns an object of class `"phylo"`. The branch lengths of this tree will be clock-like and scaled so that the root node has age 1 (or the value set by the option `scale`

Author(s)

Gangolf Jobb (`http://www.treefinder.de`) and Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

References

Sanderson, M. J. (1997) A nonparametric approach to estimating divergence times in the absence of rate constancy. *Molecular Biology and Evolution*, **14**, 1218–1231.

See Also

`ratogram`, `NPRS.criterion`.

Examples

```
# get tree
data("landplants.newick") # example tree in NH format
tree.landplants <- read.tree(text = landplants.newick)

# plot tree
tree.landplants
plot(tree.landplants, label.offset = 0.001)

# estimate chronogram
chrono.plants <- chronogram(tree.landplants)

# plot
plot(chrono.plants, label.offset = 0.001)

# value of NPRS function for our estimated chronogram
NPRS.criterion(tree.landplants, chrono.plants)
```

`chronopl` *Molecular Dating With Penalized Likelihood*

Description

This function estimates the node ages of a tree using a semi-parametric method based on penalized likelihood (Sanderson 2002). The branch lengths of the input tree are interpreted as mean numbers of substitutions (i.e., per site).

Usage

```
chronopl(phy, lambda, age.min = 1, age.max = NULL,
         node = "root", S = 1, tol = 1e-8,
         CV = FALSE, eval.max = 500, iter.max = 500, ...)
```

Arguments

`phy`	an object of class `"phylo"`.
`lambda`	value of the smoothing parameter.
`age.min`	numeric values specifying the fixed node ages (if `age.max = NULL`) or the youngest bound of the nodes known to be within an interval.
`age.max`	numeric values specifying the oldest bound of the nodes known to be within an interval.
`node`	the numbers of the nodes whose ages are given by `age.min`; `"root"` is a short-cut for the root.

`S`	the number of sites in the sequences; leave the default if branch lengths are in mean number of substitutions.
`tol`	the value below which branch lengths are considered effectively zero.
`CV`	whether to perform cross-validation.
`eval.max`	the maximal number of evaluations of the penalized likelihood function.
`iter.max`	the maximal number of iterations of the optimization algorithm.
`...`	further arguments passed to control `nlminb`.

Details

The idea of this method is to use a trade-off between a parametric formulation where each branch has its own rate, and a nonparametric term where changes in rates are minimized between contiguous branches. A smoothing parameter (lambda) controls this trade-off. If lambda = 0, then the parametric component dominates and rates vary as much as possible among branches, whereas for increasing values of lambda, the variation are smoother to tend to a clock-like model (same rate for all branches).

`lambda` must be given. The known ages are given in `age.min`, and the correponding node numbers in `node`. These two arguments must obviously be of the same length. By default, an age of 1 is assumed for the root, and the ages of the other nodes are estimated.

If `age.max = NULL` (the default), it is assumed that `age.min` gives exactly known ages. Otherwise, `age.max` and `age.min` must be of the same length and give the intervals for each node. Some node may be known exactly while the others are known within some bounds: the values will be identical in both arguments for the former (e.g., `age.min = c(10, 5), age.max = c(10, 6), node = c(15, 18)` means that the age of node 15 is 10 units of time, and the age of node 18 is between 5 and 6).

The input tree may have multichotomies. If some internal branches are of zero-length, they are collapsed (with a warning), and the returned tree will have less nodes than the input one. The presence of zero-lengthed terminal branches of results in an error since it makes little sense to have zero-rate branches.

The cross-validation used here is different from the one proposed by Sanderson (2002). Here, each tip is dropped successively and the analysis is repeated with the reduced tree: the estimated dates for the remaining nodes are compared with the estimates from the full data. For the ith tip the following is calculated:

$$\sum_{j=1}^{n-2} \frac{(t_j - t_j^{-i})^2}{t_j}$$

,

where t_j is the estimated date for the jth node with the full phylogeny, t_j^{-i} is the estimated date for the jth node after removing tip i from the tree, and n is the number of tips.

The present version uses the `nlminb` to optimise the penalized likelihood function: see its help page for details on parameters controlling the optimisation procedure.

Value

an object of class `"phylo"` with branch lengths as estimated by the function. There are three or four further attributes:

`ploglik`	the maximum penalized log-likelihood.
`rates`	the estimated rates for each branch.
`message`	the message returned by `nlminb` indicating whether the optimisation converged.
`D2`	the influence of each observation on overall date estimates (if `CV = TRUE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Sanderson, M. J. (2002) Estimating absolute rates of molecular evolution and divergence times: a penalized likelihood approach. *Molecular Biology and Evolution*, **19**, 101–109.

See Also

`chronogram`, `ratogram`, `NPRS.criterion`, `chronoMPL`

`coalescent.intervals` *Coalescent Intervals*

Description

This function extracts or generates information about coalescent intervals (number of lineages, interval lengths, interval count, total depth) from a phylogenetic tree or a list of internode distances. The input tree needs to be ultra-metric (i.e. clock-like).

Usage

```
coalescent.intervals(x)
```

Arguments

`x`	either an ultra-metric phylogenetic tree (i.e. an object of class `"phylo"`) or, alternatively, a vector of interval lengths.

Value

An object of class "coalescentIntervals" with the following entries:

lineages	A vector with the number of lineages at the start of each coalescent interval.
interval.length	A vector with the length of each coalescent interval.
interval.count	The total number of coalescent intervals.
total.depth	The sum of the lengths of all coalescent intervals.

Author(s)

Korbinian Strimmer (http://www.stat.uni-muenchen.de/~strimmer/)

See Also

branching.times, collapsed.intervals, read.tree.

Examples

```
data("hivtree.newick") # example tree in NH format
tree.hiv <- read.tree(text = hivtree.newick) # load tree

ci <- coalescent.intervals(tree.hiv) # from tree
ci

data("hivtree.table") # same tree, but in table format
ci <- coalescent.intervals(hivtree.table$size) # from vector of interval lengths
ci
```

collapse.singles *Collapse Single Nodes*

Description

This function deletes the single nodes (i.e., with a single descendant) in a tree.

Usage

```
collapse.singles(tree)
```

Arguments

tree	an object of class "phylo".

Value

an object of class "phylo".

Author(s)

Ben Bolker ⟨bolker@zoo.ufl.edu⟩

See Also

`plot.phylo`, `read.tree`

`collapsed.intervals` *Collapsed Coalescent Intervals*

Description

This function takes a `"coalescentIntervals"` objects and collapses neighbouring coalescent intervals into a single combined interval so that every collapsed interval is larger than `epsilon`. Collapsed coalescent intervals are used, e.g., to obtain the generalized skyline plot (`skyline`). For `epsilon = 0` no interval is collapsed.

Usage

```
collapsed.intervals(ci, epsilon=0)
```

Arguments

`ci`	coalescent intervals (i.e. an object of class `"coalescentIntervals"`).
`epsilon`	collapsing parameter that controls the amount of smoothing (allowed range: from `0` to `ci$total.depth`)

Details

Proceeding from the tips to the root of the tree each small interval is pooled with the neighboring interval closer to the root. If the neighboring interval is also small, then pooling continues until the composite interval is larger than `epsilon`. Note that this approach prevents the occurrence of zero-length intervals at the present. For more details see Strimmer and Pybus (2001).

Value

An object of class `"collapsedIntervals"` with the following entries:

`lineages`	A vector with the number of lineages at the start of each coalescent interval.
`interval.length`	A vector with the length of each coalescent interval.
`collapsed.interval`	A vector indicating for each coalescent interval to which collapsed interval it belongs.
`interval.count`	The total number of coalescent intervals.

`collapsed.interval.count`	The number of collapsed intervals.
`total.depth`	The sum of the lengths of all coalescent intervals.
`epsilon`	The value of the underlying smoothing parameter.

Author(s)

Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

References

Strimmer, K. and Pybus, O. G. (2001) Exploring the demographic history of DNA sequences using the generalized skyline plot. *Molecular Biology and Evolution*, **18**, 2298–2305.

See Also

`coalescent.intervals`,`skyline`.

Examples

```
data("hivtree.table") # example tree

# colescent intervals from vector of interval lengths
ci <- coalescent.intervals(hivtree.table$size)
ci

# collapsed intervals
cl1 <- collapsed.intervals(ci,0)
cl2 <- collapsed.intervals(ci,0.0119)

cl1
cl2
```

`compar.cheverud` *Cheverud's Comparative Method*

Description

This function computes the phylogenetic variance component and the residual deviation for continous characters, taking into account the phylogenetic relationships among species, following the comparative method described in Cheverud et al. (1985). The correction proposed by Rholf (2001) is used.

Usage

```
compar.cheverud(y, W, tolerance = 1e-06, gold.tol = 1e-04)
```

Arguments

`y`	A vector containing the data to analyse.
`W`	The phylogenetic connectivity matrix. All diagonal elements will be ignored.
`tolerance`	Minimum difference allowed to consider eigenvalues as distinct.
`gold.tol`	Precision to use in golden section search alogrithm.

Details

Model:

$$y = \rho W y + e$$

where e is the error term, assumed to be normally distributed. ρ is estimated by the maximum likelihood procedure given in Rohlf (2001), using a golden section search algorithm. The code of this function is indeed adapted from a MatLab code given in appendix in Rohlf's article, to correct a mistake in Cheverud's original paper.

Value

A list with the following components:

`rhohat`	The maximum likelihood estimate of ρ
`Wnorm`	The normalized version of `W`
`residuals`	Error terms (e)

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Cheverud, J. M., Dow, M. M. and Leutenegger, W. (1985) The quantitative assessment of phylogenetic constraints in comparative analyses: sexual dimorphism in body weight among primates. *Evolution*, **39**, 1335–1351.

Rohlf, F. J. (2001) Comparative methods for the analysis of continuous variables: geometric interpretations. *Evolution*, **55**, 2143–2160.

Harvey, P. H. and Pagel, M. D. (1991) *The comparative method in evolutionary biology*. Oxford University Press.

See Also

`compar.lynch`

Examples

```
### Example from Harvey and Pagel's book:
## Not run:
y<-c(10,8,3,4)
W <- matrix(c(1,1/6,1/6,1/6,1/6,1,1/2,1/2,1/6,1/2,1,1,1/6,1/2,1,1), 4)
compar.cheverud(y,W)
## End(Not run)
### Example from Rohlf's 2001 article:
W<- matrix(c(
  0,1,1,2,0,0,0,0,
  1,0,1,2,0,0,0,0,
  1,1,0,2,0,0,0,0,
  2,2,2,0,0,0,0,0,
  0,0,0,0,0,1,1,2,
  0,0,0,0,1,0,1,2,
  0,0,0,0,1,1,0,2,
  0,0,0,0,2,2,2,0
),8)
W <- 1/W
W[W == Inf] <- 0
y<-c(-0.12,0.36,-0.1,0.04,-0.15,0.29,-0.11,-0.06)
compar.cheverud(y,W)
```

`compar.gee` *Comparative Analysis with GEEs*

Description

`compar.gee` performs the comparative analysis using generalized estimating equations as described by Paradis and Claude (2002).

`drop1` tests single effects of a fitted model output from `compar.gee`.

Usage

```
compar.gee(formula, data = NULL, family = "gaussian", phy,
           scale.fix = FALSE, scale.value = 1)
## S3 method for class 'compar.gee':
drop1(object, scope, quiet = FALSE, ...)
```

Arguments

`formula`	a formula giving the model to be fitted.
`data`	the name of the data frame where the variables in `formula` are to be found; by default, the variables are looked for in the global environment.
`family`	a character string specifying the distribution assumed for the response; by default a Gaussian distribution (with link identity) is assumed (see `?family` for details on specifying the distribution, and on changing the link function).

`phy`	an object of class `"phylo"`.
`scale.fix`	logical, indicates whether the scale parameter should be fixed (TRUE) or estimated (FALSE, the default).
`scale.value`	if `scale.fix = TRUE`, gives the value for the scale (default: `scale.value = 1`).
`object`	an object of class `"compar.gee"` resulting from fitting `compar.gee`.
`scope`	<unused>.
`quiet`	a logical specifying whether to display a warning message about eventual "marginality principle violation".
`...`	further arguments to be passed to `drop1`.

Details

If a data frame is specified for the argument `data`, then its rownames are matched to the tip labels of `phy`. The user must be careful here since the function requires that both series of names perfectly match, so this operation may fail if there is a typing or syntax error. If both series of names do not match, the values in the data frame are taken to be in the same order than the tip labels of `phy`, and a warning message is issued.

If `data = NULL`, then it is assumed that the variables are in the same order than the tip labels of `phy`.

Value

`compar.gee` returns an object of class `"compar.gee"` with the following components:

`call`	the function call, including the formula.
`nobs`	the number of observations.
`coefficients`	the estimated coefficients (or regression parameters).
`residuals`	the regression residuals.
`family`	a character string, the distribution assumed for the response.
`link`	a character string, the link function used for the mean function.
`scale`	the scale (or dispersion parameter).
`W`	the variance-covariance matrix of the estimated coefficients.
`dfP`	the phylogenetic degrees of freedom (see Paradis and Claude for details on this).

`drop1` returns an object of class `"anova"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. and Claude J. (2002) Analysis of comparative data using generalized estimating equations. *Journal of theoretical Biology*, **218**, 175–185.

See Also

read.tree, pic, compar.lynch, drop1

Examples

```
### The example in Phylip 3.5c (originally from Lynch 1991)
### (the same analysis than in help(pic)...)
cat("((((Homo:0.21,Pongo:0.21):0.28,",
   "Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);",
   file = "ex.tre", sep = "\n")
tree.primates <- read.tree("ex.tre")
X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
### Both regressions... the results are quite close to those obtained
### with pic().
compar.gee(X ~ Y, phy = tree.primates)
compar.gee(Y ~ X, phy = tree.primates)
### Now do the GEE regressions through the origin: the results are quite
### different!
compar.gee(X ~ Y - 1, phy = tree.primates)
compar.gee(Y ~ X - 1, phy = tree.primates)
unlink("ex.tre") # delete the file "ex.tre"
```

---

compar.lynch *Lynch's Comparative Method*

Description

This function computes the heritable additive value and the residual deviation for continous characters, taking into account the phylogenetic relationships among species, following the comparative method described in Lynch (1991).

Usage

```
compar.lynch(x, G, eps = 1e-4)
```

Arguments

`x`	eiher a matrix, a vector, or a data.frame containing the data with species as rows and variables as columns.
`G`	a matrix that can be interpreted as an among-species correlation matrix.
`eps`	a numeric value to detect convergence of the EM algorithm.

Details

The parameter estimates are computed following the EM (expectation-maximization) algorithm. This algorithm usually leads to convergence but may lead to local optima of the likelihood function. It is recommended to run several times the function in order to detect these potential local optima. The 'optimal' value for `eps` depends actually on the range of the data and may be changed by the user in order to check the stability of the parameter estimates. Convergence occurs when the differences between two successive iterations of the EM algorithm leads to differences between both residual and additive values less than or equal to `eps`.

Value

A list with the following components:

`vare`	estimated residual variance-covariance matrix.
`vara`	estimated additive effect variance covariance matrix.
`u`	estimates of the phylogeny-wide means.
`A`	addtitive value estimates.
`E`	residual values estimates.
`lik`	logarithm of the likelihood for the entire set of observed taxon-specific mean.

Note

The present function does not perform the estimation of ancestral phentoypes as proposed by Lynch (1991). This will be implemented in a future version.

Author(s)

Julien Claude ⟨claude@isem.univ-montp2.fr⟩

References

Lynch, M. (1991) Methods for the analysis of comparative data in evolutionary biology. *Evolution*, **45**, 1065–1080.

See Also

`pic`, `compar.gee`

Examples

```
### The example in Lynch (1991)
cat("((((Homo:0.21,Pongo:0.21):0.28,",
   "Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);",
   file = "ex.tre", sep = "\n")
tree.primates <- read.tree("ex.tre")
unlink("ex.tre")
X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
compar.lynch(cbind(X, Y),
             G = vcv.phylo(tree.primates, cor = TRUE))
```

compar.ou *Ornstein–Uhlenbeck Model for Continuous Characters*

Description

This function fits an Ornstein–Uhlenbeck model giving a phylogenetic tree, and a continuous character. The user specifies the node(s) where the optimum changes. The parameters are estimated by maximum likelihood; their standard-errors are computed assuming normality of these estimates.

Usage

```
compar.ou(x, phy, node = NULL, alpha = NULL)
```

Arguments

`x`	a numeric vector giving the values of a continuous character.
`phy`	an object of class `"phylo"`.
`node`	a vector giving the number(s) of the node(s) where the parameter 'theta' (the character optimum) is assumed to change. By default there is no change (same optimum thoughout lineages).
`alpha`	the value of α to be used when fitting the model. By default, this parameter is estimated (see details).

Details

The Ornstein–Uhlenbeck (OU) process can be seen as a generalization of the Brownian motion process. In the latter, characters are assumed to evolve randomly under a random walk, that is change is equally likely in any direction. In the OU model, change is more likely towards the direction of an optimum (denoted θ) with a strength controlled by a parameter denoted α.

The present function fits a model where the optimum parameter θ, is allowed to vary throughout the tree. This is specified with the argument `node`: θ changes after each node whose number is given there. Note that the optimum changes *only* for the lineages which are descendants of this node.

Hansen (1997) recommends to not estimate α together with the other parameters. The present function allows this by giving a numeric value to the argument `alpha`. By default, this parameter is estimated, but this seems to yield very large standard-errors, thus validating Hansen's recommendation. In practice, a "poor man estimation" of α can be done by repeating the function call with different values of `alpha`, and selecting the one that minimizes the deviance (see Hansen 1997 for an example).

If `x` has names, its values are matched to the tip labels of `phy`, otherwise its values are taken to be in the same order than the tip labels of `phy`.

The user must be careful here since the function requires that both series of names perfectly match, so this operation may fail if there is a typing or syntax error. If both series of names do not match, the values in the `x` are taken to be in the same order than the tip labels of `phy`, and a warning message is issued.

Value

an object of class "`compar.ou`" which is list with the following components:

`deviance`	the deviance (= -2 * loglik).
`para`	a data frame with the maximum likelihood estimates and their standard-errors.
`call`	the function call.

Note

The inversion of the variance-covariance matrix in the likelihood function appeared as somehow problematic. The present implementation uses a Cholevski decomposition with the function `chol2inv` instead of the usual function `solve`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Hansen, T. F. (1997) Stabilizing selection and the comparative analysis of adaptation. *Evolution*, **51**, 1341–1351.

See Also

`ace`, `compar.lynch`, `corBrownian`, `corMartins`, `pic`

Examples

```
## Not run:
data(bird.orders)
### This is likely to give you estimates close to 0, 1, and 0
### for alpha, sigma^2, and theta, respectively:
compar.ou(rnorm(23), bird.orders)
### Much better with a fixed alpha:
compar.ou(rnorm(23), bird.orders, alpha = 0.1)
### Let us 'mimick' the effect of different optima
### for the two clades of birds...
x <- c(rnorm(5, 0), rnorm(18, 5))
### ... the model with two optima:
compar.ou(x, bird.orders, node = -2, alpha = .1)
### ... and the model with a single optimum:
compar.ou(x, bird.orders, node = NULL, alpha = .1)
### => Compare both models with the difference in deviances
##     with follows a chi^2 with df = 1.
## End(Not run)
```

`compute.brlen` *Branch Lengths Computation*

Description

This function computes branch lengths of a tree using different methods.

Usage

```
compute.brlen(phy, method = "Grafen", power = 1, ...)
```

Arguments

`phy`	an object of class `phylo` representing the tree.
`method`	the method to be used to compute the branch lengths; this must be one of the followings: (i) `"Grafen"` (the default), (ii) a numeric vector, or (iii) a function.
`power`	The power at which heights must be raised (see below).
`...`	further argument(s) to be passed to `method` if it is a function.

Details

Grafen's (1989) computation of branch lengths: each node is given a 'height', namely the number of leaves of the subtree minus one, 0 for leaves. Each height is scaled so that root height is 1, and then raised at power 'rho' (> 0). Branch lengths are then computed as the difference between height of lower node and height of upper node.

If one or several numeric values are provided as `method`, they are recycled if necessary. If a function is given instead, further arguments are given in place of `...` (they must be named, see examples).

Zero-length branches are not treated as multichotomies, and thus may need to be collapsed (see `di2multi`).

Value

An object of class `phylo` with branch lengths.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩ and Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Grafen, A. (1989) The phylogenetic regression. *Philosophical Transactions of the Royal society of London. Series B. Biological Sciences*, **326**, 119–157.

See Also

`read.tree` for a description of `phylo` objects, `di2multi`, `multi2di`

Examples

```
data(bird.orders)
plot(compute.brlen(bird.orders, 1))
plot(compute.brlen(bird.orders, runif, min = 0, max = 5))
layout(matrix(1:4, 2, 2))
plot(compute.brlen(bird.orders, power=1), main=expression(rho==1))
plot(compute.brlen(bird.orders, power=3), main=expression(rho==3))
plot(compute.brlen(bird.orders, power=0.5), main=expression(rho==0.5))
plot(compute.brlen(bird.orders, power=0.1), main=expression(rho==0.1))
layout(1)
```

`consensus` *Concensus Trees*

Description

Given a series of trees, this function returns the consensus tree. By default, the strict-consensus tree is computed. To get the majority-rule consensus tree, use `p = 0.5`. Any value between 0.5 and 1 can be used.

Usage

```
consensus(..., p = 1, check.labels = FALSE)
```

Arguments

`...`	either (i) a single object of class `"phylo"`, (ii) a series of such objects separated by commas, or (iii) a list containing such objects.
`p`	a numeric value between 0.5 and 1 giving the proportion for a clade to be represented in the consensus tree.
`check.labels`	a logical specifying whether to check the labels of each tree. If `FALSE` (the default), it is assumed that all trees have the same tip labels, and that they are in the same order (see details).

Details

Using (the default) `check.labels = FALSE` results in considerable decrease in computing times. This requires that (i) all trees have the same tip labels, *and* (ii) these labels are ordered similarly in all trees (in other words, the element `tip.label` are identical in all trees).

Value

an object of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

prop.part, dist.topo

cophenetic.phylo *Pairwise Distances from a Phylogenetic Tree*

Description

cophenetic.phylo computes the pairwise distances between the pairs of tips from a phylogenetic tree using its branch lengths.

dist.nodes does the same but between all nodes, internal and terminal, of the tree.

Usage

```
## S3 method for class 'phylo':
cophenetic(x)
dist.nodes(x)
```

Arguments

x	an object of class "phylo".

Value

a numeric matrix with colnames and rownames set to the names of the tips (as given by the element tip.label of the argument phy), or, in the case of dist.nodes, the numbers of the tips and the nodes (as given by the element edge).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

read.tree to read tree files in Newick format, cophenetic for the generic function

`cophyloplot`	*Plots two phylogenetic trees face to face with the links between the tips.*

Description

This function plots two trees face to face with the links if specified. It is possible to rotate the branches of each tree around the nodes by clicking.

Usage

```
cophyloplot(x, y, assoc=NULL, use.edge.length=FALSE,space=0,
       length.line=1, gap=2, type="phylogram", rotate=FALSE,
       col="red", show.tip.label=TRUE, font=3, ...)
```

Arguments

`x, y`	two objects of class `"phylo"`.
`assoc`	a matrix with 2 columns specifying the associations between the tips. If NULL, no links will be drawn.
`use.edge.length`	a logical indicating whether the branch lengths should be used to plot the trees; default is FALSE.
`space`	a positive value that specifies the distance between the two trees.
`length.line`	a positive value that specifies the length of the horizontal line associated to each taxa. Default is 1.
`gap`	a value specifying the distance between the tips of the phylogeny and the lines.
`type`	a character string specifying the type of phylogeny to be drawn; it must be one of "phylogram" (the default) or "cladogram".
`rotate`	a logical indicating whether the nodes of the phylogeny can be rotated by clicking. Default is FALSE.
`col`	a character string indicating the color to be used for the links. Default is red.
`show.tip.label`	a logical indicating whether to show the tip labels on the phylogeny (defaults to 'TRUE', i.e. the labels are shown).
`font`	an integer specifying the type of font for the labels: 1 (plain text), 2 (bold), 3 (italic, the default), or 4 (bold italic).
`...`	(unused)

Details

The aim of this function is to plot simultaneously two phylogenetic trees with associated taxa. The two trees do not necessarily have the same number of tips and more than one tip in one phylogeny can be associated with a tip in the other.

The association matrix used to draw the links has to be a matrix with two columns containing the names of the tips. One line in the matrix represents one link on the plot. The first column of the matrix has to contain tip labels of the first tree (`phy1`) and the second column of the matrix, tip labels of the second tree (`phy2`). There is no limit (low or high) for the number of lines in the matrix. A matrix with two colums and one line will give a plot with one link.

Arguments `gap`, `length.line` and `space` have to be changed to get a nice plot of the two phylogenies. Note that the function takes into account the length of the character strings corresponding to the names at the tips, so that the lines do not overwrite those names.

The `rotate` argument can be used to transform both phylogenies in order to get the more readable plot (typically by decreasing the number of crossing lines). This can be done by clicking on the nodes. The escape button or right click take back to the console.

Author(s)

Damien de Vienne ⟨damien.de-vienne@u-psud.fr⟩

See Also

`plot.phylo`, `rotate`

Examples

```
#two random trees
tree1<-rtree(40) #random tree with 40 leaves
tree2<-rtree(20) #random tree with 20 leaves

#creation of the association matrix
association<-matrix(ncol=2, nrow=40)
association[,1]<-association[,2]<-tree2$tip.label

#plot
cophyloplot(tree1, tree2, assoc=association, length.line=4, space=28, gap=3)

#plot with rotations
## Not run:
cophyloplot(tree1, tree2, assoc=association, length.line=4, space=28, gap=3, rotate=TRUE)
## End(Not run)
```

`corBlomberg` *Blomberg et al.'s Correlation Structure*

Description

The “ACDC” (accelerated/decelerated) model assumes that continuous traits evolve under a Brownian motion model which rates accelerates (if $g < 1$) or decelerates (if $g > 1$) through time. If $g = 1$, then the model reduces to a Brownian motion model.

Usage

```
corBlomberg(value, phy, form = ~1, fixed = FALSE)
## S3 method for class 'corBlomberg':
corMatrix(object, covariate = getCovariate(object),
                   corr = TRUE, ...)
## S3 method for class 'corBlomberg':
coef(object, unconstrained = TRUE, ...)
```

Arguments

`value`	the (initial) value of the parameter g.
`phy`	an object of class `"phylo"`.
`form`	(ignored).
`fixed`	a logical specifying whether `gls` should estimate γ (the default) or keep it fixed.
`object`	an (initialized) object of class `"corBlomberg"`.
`covariate`	(ignored).
`corr`	a logical value specifying whether to return the correlation matrix (the default) or the variance-covariance matrix.
`unconstrained`	a logical value. If `TRUE` (the default), the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in “natural”, possibly constrained, form.
`...`	further arguments passed to or from other methods.

Value

an object of class `"corBlomberg"`, the coefficients from an object of this class, or the correlation matrix of an initialized object of this class. In most situations, only `corBlomberg` will be called by the user.

Author(s)

Emmanuel Paradis

References

Blomberg, S. P., Garland, Jr, T., and Ives, A. R. (2003) Testing for phylogenetic signal in comparative data: behavioral traits are more labile. *Evolution*, **57**, 717–745.

---

corBrownian *Brownian Correlation Structure*

Description

Expected covariance under a Brownian model (Felsenstein 1985, Martins and Hansen 1997):

$$V_{ij} = \gamma \times t_a$$

where t_a is the distance on the phylogeny between the root and the most recent common ancestor of taxa i and j and γ is a constant.

Usage

```
corBrownian(value=1, phy, form=~1)
## S3 method for class 'corBrownian':
coef(object, unconstrained = TRUE, ...)
## S3 method for class 'corBrownian':
corMatrix(object,
                covariate = getCovariate(object), corr = TRUE, ...)
```

Arguments

`value`	The γ parameter (default to 1)
`phy`	An object of class `phylo` representing the phylogeny (with branch lengths) to consider
`object`	An (initialized) object of class `corBrownian`
`corr`	a logical value. If 'TRUE' the function returns the correlation matrix, otherwise it returns the variance/covariance matrix.
`form`	ignored for now.
`covariate`	ignored for now.
`unconstrained`	a logical value. If 'TRUE' the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If 'FALSE' the coefficients are returned in "natural", possibly constrained, form. Defaults to 'TRUE'
`...`	some methods for these generics require additional arguments. None are used in these methods.

Value

An object of class `corBrownian` or coefficient from an object of this class (actually sends `numeric(0)`!) or the correlation matrix of an initialized object of this class.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Felsenstein, J. (1985) Phylogenies and the comparative method. *American Naturalist*, **125**, 1–15.

Martins, E. P. and Hansen, T. F. (1997) Phylogenies and the comparative method: a general approach to incorporating phylogenetic information into the analysis of interspecific data. *American Naturalist*, **149**, 646–667.

See Also

`corClasses`.

`corClasses` *Phylogenetic Correlation Structures*

Description

Classes of phylogenetic correlation structures (`"corPhyl"`) available in **ape**.

corBrownian Brownian motion model (Felsenstein 1985)

corMartins The covariance matrix defined in Martins and Hansen (1997)

corGrafen The covariance matrix defined in Grafen (1989)

corPagel The covariance matrix defined in Freckelton et al. (2002)

corBlomberg The covariance matrix defined in Blomberg et al. (2003)

See the help page of each class for references and detailed description.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩, Emmanuel Paradis

See Also

`corClasses` and `gls` in the **nlme** librarie, `corBrownian`, `corMartins`, `corGrafen`, `corPagel`, `corBlomberg`

Examples

```
library(nlme)
cat("((((Homo:0.21,Pongo:0.21):0.28,",
"Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);",
file = "ex.tre", sep = "\n")
tree.primates <- read.tree("ex.tre")
X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
unlink("ex.tre") # delete the file "ex.tre"
m1 <- gls(Y~X, correlation=corBrownian(1, tree.primates))
summary(m1)
m2 <- gls(Y~X, correlation=corMartins(1, tree.primates))
summary(m2)
corMatrix(m2$modelStruct$corStruct)
m3 <- gls(Y~X, correlation=corGrafen(1, tree.primates))
summary(m3)
corMatrix(m3$modelStruct$corStruct)
```

---

corGrafen *Grafen's (1989) Correlation Structure*

Description

Grafen's (1989) covariance structure. Branch lengths are computed using Grafen's method (see `compute.brlen`). The covariance matrice is then the traditional variance-covariance matrix for a phylogeny.

Usage

```
corGrafen(value, phy, form=~1, fixed = FALSE)
## S3 method for class 'corGrafen':
coef(object, unconstrained = TRUE, ...)
## S3 method for class 'corGrafen':
corMatrix(object,
                      covariate = getCovariate(object), corr = TRUE, ...)
```

Arguments

`value`	The α parameter
`phy`	An object of class `phylo` representing the phylogeny (branch lengths are ignored) to consider
`object`	An (initialized) object of class `corGrafen`
`corr`	a logical value. If 'TRUE' the function returns the correlation matrix, otherwise it returns the variance/covariance matrix.
`fixed`	an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial value. Defaults to 'FALSE', in which case the coefficients are allowed to vary.

`form`	ignored for now.
`covariate`	ignored for now.
`unconstrained`	a logical value. If 'TRUE' the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If 'FALSE' the coefficients are returned in "natural", possibly constrained, form. Defaults to 'TRUE'
`...`	some methods for these generics require additional arguments. None are used in these methods.

Value

An object of class `corGrafen` or the rho coefficient from an object of this class or the correlation matrix of an initialized object of this class.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Grafen, A. (1989) The phylogenetic regression. *Philosophical Transactions of the Royal society of London. Series B. Biological Sciences*, **326**, 119–157.

See Also

`corClasses`, `compute.brlen`, `vcv.phylo`.

`corMartins` *Martins's (1997) Correlation Structure*

Description

Martins and Hansen's (1997) covariance structure:

$$V_{ij} = \gamma \times e^{-\alpha t_{ij}}$$

where t_{ij} is the phylogenetic distance between taxa i and j and γ is a constant.

Usage

```
corMartins(value, phy, form=~1, fixed = FALSE)
## S3 method for class 'corMartins':
coef(object, unconstrained = TRUE, ...)
## S3 method for class 'corMartins':
corMatrix(object,
                covariate = getCovariate(object), corr = TRUE, ...)
```

Arguments

`value`	The α parameter
`phy`	An object of class `phylo` representing the phylogeny (with branch lengths) to consider
`object`	An (initialized) object of class `corMartins`
`corr`	a logical value. If 'TRUE' the function returns the correlation matrix, otherwise it returns the variance/covariance matrix.
`fixed`	an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, ok kept fixed at their initial value. Defaults to 'FALSE', in which case the coefficients are allowed to vary.
`form`	ignored for now.
`covariate`	ignored for now.
`unconstrained`	a logical value. If 'TRUE' the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If 'FALSE' the coefficients are returned in "natural", possibly constrained, form. Defaults to 'TRUE'
`...`	some methods for these generics require additional arguments. None are used in these methods.

Value

An object of class `corMartins` or the alpha coefficient from an object of this class or the correlation matrix of an initialized object of this class.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Martins, E. P. and Hansen, T. F. (1997) Phylogenies and the comparative method: a general approach to incorporating phylogenetic information into the analysis of interspecific data. *American Naturalist*, **149**, 646–667.

See Also

`corClasses`.

corPagel	*Pagel's "lambda" Correlation Structure*

Description

The correlation structure from the present model is derived from the Brownian motion model by multiplying the off-diagonal elements (i.e., the covariances) by γ. The variances are thus the same than for a Brownian motion model.

Usage

```
corPagel(value, phy, form = ~1, fixed = FALSE)
## S3 method for class 'corPagel':
corMatrix(object, covariate = getCovariate(object),
                   corr = TRUE, ...)
## S3 method for class 'corPagel':
coef(object, unconstrained = TRUE, ...)
```

Arguments

`value`	the (initial) value of the parameter γ.
`phy`	an object of class `"phylo"`.
`form`	(ignored).
`fixed`	a logical specifying whether `gls` should estimate γ (the default) or keep it fixed.
`object`	an (initialized) object of class `"corPagel"`.
`covariate`	(ignored).
`corr`	a logical value specifying whether to return the correlation matrix (the default) or the variance-covariance matrix.
`unconstrained`	a logical value. If `TRUE` (the default), the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in "natural", possibly constrained, form.
`...`	further arguments passed to or from other methods.

Value

an object of class `"corPagel"`, the coefficients from an object of this class, or the correlation matrix of an initialized object of this class. In most situations, only `corPagel()` will be called by the user.

Author(s)

Emmanuel Paradis

References

Freckleton, R. P., Harvey, P. H. and M. Pagel, M. (2002) Phylogenetic analysis and comparative data: a test and review of evidence. *American Naturalist*, **160**, 712–726.

Pagel, M. (1999) Inferring the historical patterns of biological evolution. *Nature*, **401**,877–884.

`correlogram.formula` *Phylogenetic Correlogram*

Description

This function computes a correlogram from taxonomic levels.

Usage

```
correlogram.formula(formula, data = NULL, use = "all.obs")
```

Arguments

`formula`	a formula of the type `y1+..+yn ~ g1/../gn`, where the `y`'s are the data to analyse and the `g`'s are the taxonomic levels.
`data`	a data frame containing the variables specified in the formula. If `NULL`, the variables are sought in the user's workspace.
`use`	a character string specifying how to handle missing values (i.e., `NA`). This must be one of "all.obs", "complete.obs", or "pairwise.complete.obs", or any unambiguous abbrevation of these. In the first case, the presence of missing values produces an error. In the second case, all rows with missing values will be removed before computation. In the last case, missing values are removed on a case-by-case basis.

Details

See the vignette in R: `vignette("MoranI")`.

Value

An object of class `correlogram` which is a data frame with three columns:

`obs`	the computed Moran's I
`p.values`	the corresponding P-values
`labels`	the names of each level

or an object of class `correlogramList` containing a list of objects of class `correlogram` if several variables are given as response in `formula`.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩ and Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.correlogram`, `plot.correlogram`

Examples

```
data(carnivora)
### Using the formula interface:
co <- correlogram.formula(SW ~ Order/SuperFamily/Family/Genus,
      data=carnivora)
co
plot(co)
### Several correlograms on the same plot:
cos <- correlogram.formula(SW + FW ~ Order/SuperFamily/Family/Genus,
      data=carnivora)
cos
plot(cos)
```

`data.nex` *NEXUS Data Example*

Description

Example of Protein data in NEXUS format (Maddison et al., 1997). Data is written in interleaved format using a single DATA block. Original data from Rokas et al (2002).

Usage

```
data(cynipids)
```

Format

ASCII text in NEXUS format

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) NEXUS: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

Rokas, A., Nylander, J. A. A., Ronquist, F. and Stone, G. N. (2002) A maximum likelihood analysis of eight phylogenetic markers in Gallwasps (Hymenoptera: Cynipidae): implications for insect phylogenetic studies. *Molecular Phylogenetics and Evolution*, **22**, 206–219.

`dist.dna` *Pairwise Distances from DNA Sequences*

Description

This function computes a matrix of pairwise distances from DNA sequences using a model of DNA evolution. Eleven substitution models (and the raw distance) are currently available.

Usage

```
dist.dna(x, model = "K80", variance = FALSE,
         gamma = FALSE, pairwise.deletion = FALSE,
         base.freq = NULL, as.matrix = FALSE)
```

Arguments

`x`	a matrix or a list containing the DNA sequences.
`model`	a character string specifying the evlutionary model to be used; must be one of `"raw"`, `"JC69"`, `"K80"` (the default), `"F81"`, `"K81"`, `"F84"`, `"BH87"`, `"T92"`, `"TN93"`, `"GG95"`, `"logdet"`, or `"paralin"`.
`variance`	a logical indicating whether to compute the variances of the distances; defaults to `FALSE` so the variances are not computed.
`gamma`	a value for the gamma parameter which is possibly used to apply a gamma correction to the distances (by default `gamma = FALSE` so no correction is applied).
`pairwise.deletion`	a logical indicating whether to delete the sites with missing data in a pairwise way. The default is to delete the sites with at least one missing data for all sequences.
`base.freq`	the base frequencies to be used in the computations (if applicable, i.e. if `method = "F84"`). By default, the base frequencies are computed from the whole sample of sequences.
`as.matrix`	a logical indicating whether to return the results as a matrix. The default is to return an object of class dist.

Details

The molecular evolutionary models available through the option `model` have been extensively described in the literature. A brief description is given below; more details can be found in the References.

"raw" This is simply the proportion of sites that differ between each pair of sequences. This may be useful to draw "saturation plots". The options `variance` and `gamma` have no effect, but `pairwise.deletion` can.

"JC69" This model was developed by Jukes and Cantor (1969). It assumes that all substitutions (i.e. a change of a base by another one) have the same probability. This probability is the same for all sites along the DNA sequence. This last assumption can be relaxed by assuming that the substition rate varies among site following a gamma distribution which parameter must be given by the user. By default, no gamma correction is applied. Another assumption is that the base frequencies are balanced and thus equal to 0.25.

"K80" The distance derived by Kimura (1980), sometimes referred to as "Kimura's 2-parameters distance", has the same underlying assumptions than the Jukes–Cantor distance except that two kinds of substitutions are considered: transitions (A <-> G, C <-> T), and transversions (A <-> C, A <-> T, C <-> G, G <-> T). They are assumed to have different probabilities. A transition is the substitution of a purine (C, T) by another one, or the substitution of a pyrimidine (A, G) by another one. A transversion is the substitution of a purine by a pyrimidine, or vice-versa. Both transition and transversion rates are the same for all sites along the DNA sequence. Jin and Nei (1990) modified the Kimura model to allow for variation among sites following a gamma distribution. Like for the Jukes–Cantor model, the gamma parameter must be given by the user. By default, no gamma correction is applied.

"F81" Felsenstein (1981) generalized the Jukes–Cantor model by relaxing the assumption of equal base frequencies. The formulae used in this function were taken from McGuire et al. (1999).

"K81" Kimura (1981) generalized his model (Kimura 1980) by assuming different rates for two kinds of transversions: A <-> C and G <-> T on one side, and A <-> T and C <-> G on the other. This is what Kimura called his "three substitution types model" (3ST), and is sometimes referred to as "Kimura's 3-parameters distance".

"F84" This model generalizes K80 by relaxing the assumption of equal base frequencies. It was first introduced by Felsenstein in 1984 in Phylip, and is fully described by Felsenstein and Churchill (1996). The formulae used in this function were taken from McGuire et al. (1999).

"BH87" Barry and Hartigan (1987) developed a distance based on the observed proportions of changes among the four bases. This distance is not symmetric.

"T92" Tamura (1992) generalized the Kimura model by relaxing the assumption of equal base frequencies. This is done by taking into account the bias in G+C content in the sequences. The substitution rates are assumed to be the same for all sites along the DNA sequence.

"TN93" Tamura and Nei (1993) developed a model which assumes distinct rates for both kinds of transition (A <-> G versus C <-> T), and transversions. The base frequencies are not assumed to be equal and are estimated from the data. A gamma correction of the inter-site variation in substitution rates is possible.

"GG95" Galtier and Gouy (1995) introduced a model where the G+C content may change through time. Different rates are assumed for transitons and transversions.

"logdet" The Log-Det distance, developed by Lockhart et al. (1994), is related to BH87. However, this distance is symmetric.

"paralin" Lake (1994) developed the paralinear distance which can be viewed as another variant of the Barry–Hartigan distance.

Value

an object of class dist (by default), or a numeric matrix if `as.matrix = TRUE`. If `model = "BH87"`, a numeric matrix is returned because the Barry–Hartigan distance is not symmetric.

If `variance = TRUE` an attribute called `"variance"` is given to the returned object.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Barry, D. and Hartigan, J. A. (1987) Asynchronous distance between homologous DNA sequences. *Biometrics*, **43**, 261–276.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17**, 368–376.

Felsenstein, J. and Churchill, G. A. (1996) A Hidden Markov model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution*, **13**, 93–104.

Galtier, N. and Gouy, M. (1995) Inferring phylogenies from DNA sequences of unequal base compositions. *Proceedings of the National Academy of Sciences USA*, **92**, 11317–11321.

Jukes, T. H. and Cantor, C. R. (1969) Evolution of protein molecules. in *Mammalian Protein Metabolism*, ed. Munro, H. N., pp. 21–132, New York: Academic Press.

Kimura, M. (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, **16**, 111–120.

Kimura, M. (1981) Estimation of evolutionary distances between homologous nucleotide sequences. *Proceedings of the National Academy of Sciences USA*, **78**, 454–458.

Jin, L. and Nei, M. (1990) Limitations of the evolutionary parsimony method of phylogenetic analysis. *Molecular Biology and Evolution*, **7**, 82–102.

Lake, J. A. (1994) Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances. *Proceedings of the National Academy of Sciences USA*, **91**, 1455–1459.

Lockhart, P. J., Steel, M. A., Hendy, M. D. and Penny, D. (1994) Recovering evolutionary trees under a more realistic model of sequence evolution. *Molecular Biology and Evolution*, **11**, 605–602.

McGuire, G., Prentice, M. J. and Wright, F. (1999). Improved error bounds for genetic distances from DNA sequences. *Biometrics*, **55**, 1064–1070.

Tamura, K. (1992) Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G + C-content biases. *Molecular Biology and Evolution*, **9**, 678–687.

Tamura, K. and Nei, M. (1993) Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution*, **10**, 512–526.

See Also

`read.GenBank`, `read.dna`, `write.dna`, `DNAbin`, `dist.gene`, `cophenetic.phylo`, `dist`

`dist.gene` *Pairwise Distances from Genetic Data*

Description

These functions compute a matrix of distances between pairs of individuals from a matrix or a data frame of genetic data.

Usage

```
dist.gene(x, method = "pairwise", variance = FALSE)
dist.gene.pairwise(x, variance = FALSE)
dist.gene.percentage(x, variance = FALSE)
```

Arguments

`x`	a matrix or a data frame.
`method`	a character string specifying the method used to compute the distances; only two choices are available: `"pairwise"`, and `"percentage"`.
`variance`	a logical, indicates whether the variance of the distances should be returned (default to FALSE).

Details

This function is meant to be very general and accepts different kinds of data (alleles, haplotypes, DNA sequences, and so on). The rows of the data matrix represent the individuals, and the columns the loci.

In the case of the pairwise method, the distance d between two individuals is the number of loci for which they differ, and the associated variance is $d(L - d)/L$, where L is the number of loci.

In the case of the percentage method, this distance is divided by L, and the associated variance is $d(1 - d)/L$.

For more elaborate distances with DNA sequences, see the function `dist.dna`.

Value

either a numeric matrix with possibly the names of the individuals (as given by the rownames of the argument `x`) as colnames and rownames (if `variance = FALSE`, the default), or a list of two matrices names `distances` and `variance`, respectively (if `variance = TRUE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`dist.dna`, `cophenetic.phylo`

`dist.topo` *Topological Distances Between Two Trees*

Description

This function computes the topological distance between two phylogenetic trees using different methods.

Usage

```
dist.topo(x, y, method = "PH85")
```

Arguments

`x`	an object of class `"phylo"`.
`y`	an object of class `"phylo"`.
`method`	a character string giving the method to be used: either `"PH85"`, or `"BHV01"`.

Details

Two methods are available: the one by Penny and Hendy (1985), and the one by Billera et al. (2001).

The topological distance is defined as twice the number of internal branches defining different bipartitions of the tips (Penny and Hendy 1985). Rzhetsky and Nei (1992) proposed a modification of the original formula to take multifurcations into account.

Billera et al. (2001) developed a distance from the geometry of a tree space. The distance between two trees can be seen as the sum of the branch lengths that need be erased to have two similar trees.

Value

a single numeric value.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Billera, L. J., Holmes, S. P. and Vogtmann, K. (2001) Geometry of the space of phylogenetic trees. *Advances in Applied Mathematics*, **27**, 733–767.

Nei, M. and Kumar, S. (2000) *Molecular evolution and phylogenetics*. Oxford: Oxford University Press.

Penny, D. and Hendy, M. D. (1985) The use of tree comparison metrics. *Systemetic Zoology*, **34**, 75–82.

Rzhetsky, A. and Nei, M. (1992) A simple method for estimating and testing minimum-evolution trees. *Molecular Biology and Evolution*, **9**, 945–967.

See Also

`read.tree` to read tree files in Newick format, `cophenetic.phylo`, `prop.part`

Examples

```
ta <- rtree(30)
tb <- rtree(30)
dist.topo(ta, ta) # = 0
dist.topo(ta, tb) # This is unlikely to be 0 !
```

`diversi.gof` *Tests of Constant Diversification Rates*

Description

This function computes two tests of the distribution of branching times using the Cramér–von Mises and Anderson–Darling goodness-of-fit tests. By default, it is assumed that the diversification rate is constant, and an exponential distribution is assumed for the branching times. In this case, the expected distribution under this model is computed with a rate estimated from the data. Alternatively, the user may specify an expected cumulative density function (`z`): in this case, `x` and `z` must be of the same length. See the examples for how to compute the latter from a sample of expected branching times.

Usage

```
diversi.gof(x, null = "exponential", z = NULL)
```

Arguments

`x`	a numeric vector with the branching times.
`null`	a character string specifying the null distribution for the branching times. Only two choices are possible: either `"exponential"`, or `"user"`.
`z`	used if `null = "user"`; gives the expected distribution under the model.

Details

The Cramér–von Mises and Anderson–Darling tests compare the empirical density function (EDF) of the observations to an expected cumulative density function. By contrast to the Kolmogorov–Smirnov test where the greatest difference between these two functions is used, in both tests all differences are taken into account.

The distributions of both test statistics depend on the null hypothesis, and on whether or not some parameters were estimated from the data. However, these distributions are not known precisely and critical values were determined by Stephens (1974) using simulations. These critical values were used for the present function.

Value

A NULL value is returned, the results are simply printed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. (1998) Testing for constant diversification rates using molecular phylogenies: a general approach based on statistical tests for goodness of fit. *Molecular Biology and Evolution*, **15**, 476–479.

Stephens, M. A. (1974) EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, **69**, 730–737.

See Also

`branching.times`, `diversi.time` `ltt.plot`, `birthdeath`, `yule`, `yule.cov`

Examples

```
data(bird.families)
x <- branching.times(bird.families)
### suppose we have a sample of expected branching times `y';
### for simplicity, take them from a uniform distribution:
y <- runif(500, 0, max(x) + 1) # + 1 to avoid A2 = Inf
### now compute the expected cumulative distribution:
x <- sort(x)
N <- length(x)
ecdf <- numeric(N)
for (i in 1:N) ecdf[i] <- sum(y <= x[i])/500
### finally do the test:
diversi.gof(x, "user", z = ecdf)
```

`diversi.time` *Analysis of Diversification with Survival Models*

Description

This functions fits survival models to a set of branching times, some of them may be known approximately (censored). Three models are fitted, Model A assuming constant diversification, Model B assuming that diversification follows a Weibull law, and Model C assuming that diversification changes with a breakpoint at time ‘Tc’. The models are fitted by maximum likelihood.

Usage

```
diversi.time(x, census = NULL, censoring.codes = c(1, 0), Tc = NULL)
```

Arguments

`x`	a numeric vector with the branching times.
`census`	a vector of the same length than 'x' used as an indicator variable; thus, it must have only two values, one coding for accurately known branching times, and the other for censored branching times. This argument can be of any mode (numeric, character, logical), or can even be a factor.
`censoring.codes`	a vector of length two giving the codes used for `census`: by default 1 (accurately known times) and 0 (censored times). The mode must be the same than the one of `census`.
`Tc`	a single numeric value specifying the break-point time to fit Model C. If none is provided, then it is set arbitrarily to the mean of the analysed branching times.

Details

The principle of the method is to consider each branching time as an event: if the branching time is accurately known, then it is a failure event; if it is approximately knwon then it is a censoring event. An analogy is thus made between the failure (or hazard) rate estimated by the survival models and the diversification rate of the lineage. Time is here considered from present to past.

Model B assumes a monotonically changing diversification rate. The parameter that controls the change of this rate is called beta. If beta is greater than one, then the diversification rate decreases through time; if it is lesser than one, the the rate increases through time. If beta is equal to one, then Model B reduces to Model A.

Value

A NULL value is returned, the results are simply printed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. (1997) Assessing temporal variations in diversification rates from phylogenies: estimation and hypothesis testing. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **264**, 1141–1147.

See Also

`branching.times`, `diversi.gof` `ltt.plot`, `birthdeath`, `bd.ext`, `yule`, `yule.cov`

`drop.tip` *Remove Tips in a Phylogenetic Tree*

Description

This function removes the terminal branches of a phylogenetic tree, possibly removing the corresponding internal branches.

Usage

```
drop.tip(phy, tip, trim.internal = TRUE, subtree = FALSE,
         root.edge = 0)
```

Arguments

`phy`	an object of class `"phylo"`.
`tip`	a vector of mode numeric or character specifying the tips to delete.
`trim.internal`	a logical specifying whether to delete the corresponding internal branches.
`subtree`	a logical specifying whether to output in the tree how many tips have been deleted and where.
`root.edge`	an integer giving the number of internal branches to be used to build the new root edge. This has no effect if `trim.internal = FALSE`.

Details

The argument `tip` can be either character or numeric. In the first case, it gives the labels of the tips to be deleted; in the second case the numbers of these labels in the vector `phy$tip.label` are given.

If `trim.internal = FALSE`, the new tips are given `"NA"` as labels, unless there are node labels in the tree in which case they are used.

If `subtree = TRUE`, the returned tree has one or several terminal branches indicating how many tips have been removed (with a label `"[x_tips]"`). This is done for as many monophyletic groups that have been deleted.

Note that `subtree = TRUE` implies `trim.internal = TRUE`.

To undestand how the option `root.edge` works, see the examples below.

Value

an object of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`bind.tree`, `root`

Examples

```
data(bird.families)
tip <- c(
"Eopsaltriidae", "Acanthisittidae", "Pittidae", "Eurylaimidae",
"Philepittidae", "Tyrannidae", "Thamnophilidae", "Furnariidae",
"Formicariidae", "Conopophagidae", "Rhinocryptidae", "Climacteridae",
"Menuridae", "Ptilonorhynchidae", "Maluridae", "Meliphagidae",
"Pardalotidae", "Petroicidae", "Irenidae", "Orthonychidae",
"Pomatostomidae", "Laniidae", "Vireonidae", "Corvidae",
"Callaeatidae", "Picathartidae", "Bombycillidae", "Cinclidae",
"Muscicapidae", "Sturnidae", "Sittidae", "Certhiidae",
"Paridae", "Aegithalidae", "Hirundinidae", "Regulidae",
"Pycnonotidae", "Hypocoliidae", "Cisticolidae", "Zosteropidae",
"Sylviidae", "Alaudidae", "Nectariniidae", "Melanocharitidae",
"Paramythiidae","Passeridae", "Fringillidae")
plot(drop.tip(bird.families, tip))
plot(drop.tip(bird.families, tip, trim.internal = FALSE))
data(bird.orders)
plot(drop.tip(bird.orders, 6:23, subtree = TRUE), font = 1)
plot(drop.tip(bird.orders, c(1:5, 20:23), subtree = TRUE), font = 1)

### Examples of the use of `root.edge'
tr <- read.tree(text = "(A:1,(B:1,(C:1,(D:1,E:1):1):1):1):1;")
drop.tip(tr, c("A", "B"), root.edge = 0) # = (C:1,(D:1,E:1):1);
drop.tip(tr, c("A", "B"), root.edge = 1) # = (C:1,(D:1,E:1):1):1;
drop.tip(tr, c("A", "B"), root.edge = 2) # = (C:1,(D:1,E:1):1):2;
drop.tip(tr, c("A", "B"), root.edge = 3) # = (C:1,(D:1,E:1):1):3;
```

`evolve.phylo` *Ancestral Character Simulation*

Description

Simulate the (independent) evolution of one or several continuous characters along a given phylogenetic tree under a homogeneous Brownian model.

Usage

```
evolve.phylo(phy, value, var)
```

Arguments

`phy`	an object of class 'phylo' with branch lengths.
`value`	ancestral states, one by character. The (optional) names of this vector will be used as character names.
`var`	the variance of each character.

Details

Let x be a continuous character. If it evolves according to a Brownian model, its value at time t follows a normal law with mean x0 and variance t*sigma_x, where x0 is the value of the character at time 0, and sigma_x is the 'inner' variance of the character. The evolution of a continuous character is performed by letting the character evolve on each branch, from its ancestral root state. The final state of a branch is the ancestral states of the daughter branches, and so on.

Value

An object of class 'ancestral', inheriting from the 'phylo' class. The following components are added:

`node.character`	a data.frame with node ids as rownames and one column by character, containing all the inner node values for each character.
`tip.character`	a data.frame with tip ids as rownames and one column by character, containing all the tip values for each character.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

See Also

`plot.ancestral`, `ace`

Examples

```
data(bird.orders)
x <- rep(0, 5)
names(x) <- c("A", "B", "C", "D", "E")
anc1 <- evolve.phylo(bird.orders, x, 1)
anc2 <- evolve.phylo(bird.orders, x, 1)
cor(anc1$tip.character, anc2$tip.character)
```

`FastME` *Tree Estimation Based on the Minimum Evolution Algorithm*

Description

The two FastME functions (balanced and OLS) perform the Minimum Evolution algorithm of Desper and Gascuel (2002).

Usage

```
   fastme.bal(X, nni = TRUE)
   fastme.ols(X, nni = TRUE)
```

Arguments

`X`	a distance matrix; may be an object of class `"dist"`.
`nni`	a boolean value; TRUE to do NNIs (default).

Value

an object of class `"phylo"`.

Author(s)

original C code by Richard Desper; adapted and ported to R by Vincent Lefort ⟨vincent.lefort@lirmm.fr⟩

References

Desper, R. and Gascuel, O. (2002) Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, **9(5)**, 687–705.

See Also

`nj`, `bionj`, `write.tree`, `read.tree`, `dist.dna`, `mlphylo`

Examples

```
### From Saitou and Nei (1987, Table 1):
x <- c(7, 8, 11, 13, 16, 13, 17, 5, 8, 10, 13,
       10, 14, 5, 7, 10, 7, 11, 8, 11, 8, 12,
       5, 6, 10, 9, 13, 8)
M <- matrix(0, 8, 8)
M[row(M) > col(M)] <- x
M[row(M) < col(M)] <- x
rownames(M) <- colnames(M) <- 1:8
tr <- fastme.bal(M)
plot(tr, "u")
### a less theoretical example
data(woodmouse)
trw <- fastme.bal(dist.dna(woodmouse))
plot(trw)
```

`gammaStat` *Gamma-Statistic of Pybus and Harvey*

Description

This function computes the gamma-statistic which summarizes the information contained in the inter-node intervals of a phylogeny. It is assumed that the tree is ultrametric. Note that the function does not check that the tree is effectively ultrametric, so if it is not, the returned result may not be meaningful.

Usage

```
gammaStat(phy)
```

Arguments

`phy`	an object of class `"phylo"`.

Details

The gamma-statistic is a summary of the information contained in the inter-node intervals of a phylogeny; it follows, under the assumption that the clade diversified with constant rates, a normal distribution with mean zero and standard-deviation unity (Pybus and Harvey 2000). Thus, the null hypothesis that the clade diversified with constant rates may be tested with `2*(1 - pnorm(abs(gammaStat(phy))))` for a two-tailed test, or `1 - pnorm(abs(gammaStat(phy)))` for a one-tailed test, both returning the corresponding P-value.

Value

a numeric vector of length one.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Pybus, O. G. and Harvey, P. H. (2000) Testing macro-evolutionary models using incomplete molecular phylogenies. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **267**, 2267–2272.

See Also

`branching.times`, `ltt.plot`, `skyline`

`heterozygosity` *Heterozygosity at a Locus Using Gene Frequencies*

Description

This function computes the mean heterozygosity from gene frequencies, and returns optionally the associated variance.

Usage

```
heterozygosity(x, variance = FALSE)
H(x, variance = FALSE)
```

Arguments

`x`	a vector or a factor.
`variance`	a logical indicating whether the variance of the estimated heterozygosity should be returned (`TRUE`), the default being `FALSE`.

Details

The argument `x` can be either a factor or a vector. If it is a factor, then it is taken to give the individual alleles in the population. If it is a numeric vector, then its values are taken to be the numbers of each allele in the population. If it is a non-numeric vector, it is a coerced as a factor.

The mean heterozygosity is estimated with:

$$\hat{H} = \frac{n}{n-1}\left(1 - \sum_{i=1}^{k} p_i^2\right)$$

where n is the number of genes in the sample, k is the number of alleles, and p_i is the observed (relative) frequency of the allele i.

Value

a numeric vector of length one with the estimated mean heterozygosity (the default), or of length two if the variance is returned `variance = TRUE`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Nei, M. (1987) *Molecular evolutionary genetics*. New York: Columbia University Press.

See Also

`theta.s`

`hivtree` *Phylogenetic Tree of 193 HIV-1 Sequences*

Description

This data set describes an estimated clock-like phylogeny of 193 HIV-1 group M sequences sampled in the Democratic Republic of Congo.

Usage

```
data(hivtree.newick)
data(hivtree.table)
```

Format

`hivtree.newick` is a string with the tree in Newick format. The data frame `hivtree.table` contains the corresponding internode distances.

Source

This is a data example from Strimmer and Pybus (2001).

References

Strimmer, K. and Pybus, O. G. (2001) Exploring the demographic history of DNA sequences using the generalized skyline plot. *Molecular Biology and Evolution*, **18**, 2298–2305.

Examples

```
# example tree in NH format (a string)
data("hivtree.newick")
hivtree.newick

# generate file "hivtree.phy" in working directory
cat(hivtree.newick, file = "hivtree.phy", sep = "\n")
tree.hiv <- read.tree("hivtree.phy") # load tree
unlink("hivtree.phy") # delete the file "hivtree.phy"

plot(tree.hiv)

# table with list of internode distances
data("hivtree.table")
hivtree.table

# construct coalescence intervals
ci <- coalescent.intervals(tree.hiv) # from tree
ci <- coalescent.intervals(hivtree.table$size) #from intervals
ci
```

`howmanytrees` *Calculate Numbers of Phylogenetic Trees*

Description

This function calculates the number of possible phylogenetic trees for a given number of tips.

Usage

```
howmanytrees(n, rooted = TRUE, binary = TRUE,
               labeled = TRUE, detail = FALSE)
```

Arguments

`n`	a positive numeric integer giving the number of tips.
`rooted`	a logical indicating whether the trees are rooted (default is `TRUE`).
`binary`	a logical indicating whether the trees are bifurcating (default is `TRUE`).
`labeled`	a logical indicating whether the trees have tips labeled (default is `TRUE`).
`detail`	a logical indicating whether the eventual intermediate calculations should be returned (default is `FALSE`). This applies only for the multifurcating trees, and the bifurcating, rooted, unlabeled trees (aka tree shapes).

Details

In the cases of labeled binary trees, the calculation is done directly and a single numeric value is returned.

For multifurcating trees, and bifurcating, rooted, unlabeled trees, the calculation is done iteratively for 1 to `n` tips. Thus the user can print all the intermediate values if `detail = TRUE`, or only a single value if `detail = FALSE` (the default).

For multifurcating trees, if `detail = TRUE`, a matrix is returned with the number of tips as rows (named from `1` to `n`), and the number of nodes as columns (named from `1` to `n - 1`). For bifurcating, rooted, unlabeled trees, a vector is returned with names equal to the number of tips (from `1` to `n`).

The number of unlabeled trees (aka tree shapes) can be computed only for the rooted binary cases.

Note that if an infinite value (`Inf`) is returned this does not mean that there is an infinite number of trees (this cannot be if the number of tips is finite), but that the calculation is beyond the limits of the computer.

Value

a single numeric value, or in the case where `detail = TRUE` is used, a named vector or matrix.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Felsenstein, J. (2004) *Inferring phylogenies*. Sunderland: Sinauer Associates.

Examples

```
### Table 3.1 in Felsenstein 2004:
for (i in c(1:20, 30, 40, 50))
  cat(paste(i, howmanytrees(i), sep = "\t"), sep ="\n")
### Table 3.6:
howmanytrees(8, binary = FALSE, detail = TRUE)
```

identify.phylo *Graphical Identification of Nodes and Tips*

Description

This function allows to identify a clade on a plotted tree by clicking on the plot with the mouse. The tree, specified in the argument `x`, must be plotted beforehand.

Usage

```
## S3 method for class 'phylo':
identify(x, nodes = TRUE, tips = FALSE,
                  labels = FALSE, ...)
```

Arguments

`x`	an object of class `"phylo"`.
`nodes`	a logical specifying whether to identify the node.
`tips`	a logical specifying whether to return the tip information.
`labels`	a logical specifying whether to return the labels; by default only the numbers are returned.
`...`	further arguments to be passed to or from other methods.

Details

By default, the clade is identified by its number as found in the 'edge' matrix of the tree. If `tips = TRUE`, the tips descending from the identified node are returned, possibly together with the node. If `labels = TRUE`, the labels are returned (if the tree has no node labels, then the node numbered is returned).

The node is identified by the shortest distance where the click occurs. If the click occurs close to a tip, the function returns its information.

Value

A list with one or two vectors named `"tips"` and/or `"nodes"` with the identification of the tips and/or of the nodes.

Note

This function does not add anything on the plot, but it can be wrapped with, e.g., `nodelabels` (see example), or its results can be sent to, e.g., `drop.tip`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.phylo`, `nodelabels`, `identify` for the generic function

Examples

```
## Not run:
tr <- rtree(20)
f <- function(col) {
    o <- identify(tr)
    nodelabels(node=o$nodes, pch = 19, col = col)
}
plot(tr)
f("red") # click close to a node
f("green")
## End(Not run)
```

`is.binary.tree` *Test for Binary Tree*

Description

This function tests whether a phylogenetic tree is binary, i.e. whether every node (including the root node) has exactly two descendant nodes. In this case the number of edges in the tree equals 2 n - 2 where n is the number of taxa (tips) in the tree.

Usage

```
is.binary.tree(phy)
```

Arguments

`phy`	phylogenetic tree (i.e. an object of class `"phylo"`).

Value

`is.binary.tree` returns `TRUE` if `tree` is a fully binary phylogenetic tree, otherwise it returns `FALSE`.

Author(s)

Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

See Also

`read.tree`, `is.ultrametric`, `multi2di`

Examples

```
data("hivtree.newick") # example tree in NH format
tree.hiv <- read.tree(text = hivtree.newick) # load tree

is.binary.tree(tree.hiv)

plot(tree.hiv)

unlink("hivtree.phy") # delete the file "hivtree.phy"
```

`is.ultrametric` *Test if a Tree is Ultrametric*

Description

This function computes the distances from each tip to the root: if the variance of these distances is null, the tree is considered as ultrametric.

Usage

```
is.ultrametric(phy, tol = .Machine$double.eps^0.5)
```

Arguments

`phy`	an object of class `"phylo"`.
`tol`	a numeric >= 0, variation below this value are considered non-significant (see details).

Details

The default value for `tol` is based on the numerical characteristics of the machine R is running on.

Value

a logical: `TRUE` if the tree is ultrametric, `FALSE` otherwise.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`is.binary.tree`, `.Machine`

klastorin — *Klastorin's (1982) method for classifying genes as suggested by Misawa and Tajima (2000)*

Description

The function `klastorin` uses the method by Klastorin's (1982) as suggested by Misawa and Tajima (2000) for identifying groups within gene trees.

Usage

```
klastorin(phy)
```

Arguments

`phy`	a phylogenetic tree, i.e. an object of class `"phy"`. The root of the tree should make sense biologically.

Value

A vector indication the class affiliation for each sequence/taxon in the tree.

Author(s)

Gangolf Jobb (`http://www.treefinder.de`)

References

Klastorin T.D. (1982) An alternative method for hospital partition determination using hierarchical cluster analysis. *Operations Research* **30**,1134–1147.

Misawa, K. (2000) A simple method for classifying genes and a bootstrap test for classifications. *Molecular Biology and Evolution*, **17**, 1879–1884.

See Also

`opsin`.

Examples

```
# find groups in landplant tree
data("landplants.newick")
tree1 <- read.tree(text = landplants.newick)
plot(tree1, label.offset = 0.001)
klastorin(tree1)
tree1$tip.label

# find groups in opsin tree
data("opsin.newick")
```

```
tree2 <- read.tree(text = opsin.newick)
plot(tree2,label.offset = 0.01)
groups <- klastorin(tree2)
groups
tree2$tip.label[groups==1]
tree2$tip.label[groups==2]
tree2$tip.label[groups==3]
tree2$tip.label[groups==4]
tree2$tip.label[groups==5]
```

`ladderize` *Ladderize a Tree*

Description

This function reorganizes the internal structure of the tree to get the ladderized effect when plotted.

Usage

```
ladderize(phy, right = TRUE)
```

Arguments

`phy`	an object of class `"phylo"`.
`right`	a logical specifying whether the smallest clade is on the right-hand side (when the tree is plotted upwards), or the opposite (if `FALSE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.phylo`, `reorder.phylo`

Examples

```
tr <- rcoal(50)
layout(matrix(1:4, 2, 2))
plot(tr, main = "normal")
plot(ladderize(tr), main = "right-ladderized")
plot(ladderize(tr, FALSE), main = "left-ladderized")
layout(matrix(1, 1))
```

`landplants` *Gene Tree of 36 Landplant rbcL Sequences*

Description

This data set describes a gene tree estimated from 36 landplant *rbc*L sequences.

Usage

```
data(landplants.newick)
```

Format

`landplants.newick` is a string with the tree in Newick format.

Source

This tree is described in Sanderson (1997) and is also a data example in the software package r8s (`http://ginger.ucdavis.edu/r8s/`).

References

Sanderson, M. J. (1997) A nonparametric approach to estimating divergence times in the absence of rate constancy. *Molecular Biology and Evolution*, **14**, 1218–1231.

See Also

`chronogram`, `ratogram`, `NPRS.criterion`.

Examples

```
# example tree in NH format (a string)
data("landplants.newick")
landplants.newick

# get corresponding phylo object
tree.landplants <- read.tree(text = landplants.newick)

# plot tree
plot(tree.landplants, label.offset = 0.001)
```

`ltt.plot` *Lineages Through Time Plot*

Description

These functions plot, on the current graphical device, the minimum numbers of lineages through time from phylogenetic trees.

Usage

```
ltt.plot(phy, xlab = "Time", ylab = "N", ...)
ltt.lines(phy, ...)
mltt.plot(phy, ..., dcol = TRUE, dlty = FALSE, legend = TRUE,
          xlab = "Time", ylab = "N", log = "")
```

Arguments

`phy`	an object of class `"phylo"`; this could be an object of class `"multiPhylo"` in the case of `mltt.plot`.
`xlab`	a character string (or a variable of mode character) giving the label for the x-axis (default is "Time").
`ylab`	idem for the y-axis (default is "N").
`...`	in the cases of `ltt.plot()` and `ltt.lines()` these are further (graphical) arguments to be passed to `plot()` or `lines()`, respectively (see `Details:` on how to transform the axes); in the case `mltt.plot()` these are additional trees to be plotted (see `Details:`).
`dcol`	a logical specifying whether the different curves should be differentiated with colors (default is `TRUE`).
`dlty`	a logical specifying whether the different curves should be differentiated with patterns of dots and dashes (default is `FALSE`).
`legend`	a logical specifying whether a legend should be plotted.
`log`	a character string specifying which axis(es) to be log-transformed; must be one of the followings: `""`, `"x"`, `"y"`, or `"xy"`.

Details

`ltt.plot` does a simple lineages through time (LTT) plot. Additional arguments (`...`) may be used to change, for instance, the limits on the axes (with `xlim` and/or `ylim`) or other graphical settings (`col` for the color, `lwd` for the line thickness, `lty` for the line type may be useful; see `par` for an exhaustive listing of graphical parameters). The y-axis can be log-transformed by adding the following option: `log = "y"`.

`ltt.lines` adds a LTT curve to an existing plot. Additional arguments (`...`) may be used to change the settings of the added line. Of course, the settings of the already existing LTT plot cannot be altered this way.

`mltt.plot` does a multiple LTT plot taking as arguments one or several trees. These trees may be given as objects of class `"phylo"` (single trees) or `"multiPhylo"` (multiple trees). Any number of objects may be given. This function is mainly for exploratory analyses with the advantages that the axes are set properly to view all lines, and the legend is plotted by default. For more flexible settings of line drawings, it is probably better to combine `ltt.plot()` with successive calls of `ltt.lines()` (see `Examples:`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Harvey, P. H., May, R. M. and Nee, S. (1994) Phylogenies without fossils. *Evolution*, **48**, 523–529.

Nee, S., Holmes, E. C., Rambaut, A. and Harvey, P. H. (1995) Inferring population history from molecular phylogenies. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, **349**, 25–31.

See Also

`skyline`, `branching.times`, `birthdeath`, `bd.ext`, `yule.cov` `plot` for the basic plotting function in R

Examples

```
data(bird.families)
data(bird.orders)
opar <- par(mfrow = c(2, 1))
ltt.plot(bird.families)
title("Lineages Through Time Plot of the Bird Families")
ltt.plot(bird.families, log = "y")
title(main = "Lineages Through Time Plot of the Bird Families",
      sub = "(with logarithmic transformation of the y-axis)")
par(opar)
### to plot the tree and the LTT plot together
layout(matrix(1:4, 2, 2))
plot(bird.families, show.tip.label = FALSE)
ltt.plot(bird.families, main = "Bird families")
plot(bird.orders, show.tip.label = FALSE)
ltt.plot(bird.orders, main = "Bird orders")
layout(matrix(1))
mltt.plot(bird.families, bird.orders)
### Generates 10 random trees with 23 tips:
TR <- replicate(10, rcoal(23), FALSE)
### Give names to each tree:
names(TR) <- paste("random tree", 1:10)
### And specify the class of the list so that mltt.plot()
### does not trash it!
class(TR) <- "multiPhylo"
mltt.plot(TR, bird.orders)
### And now for something (not so) completely different:
```

```
ltt.plot(bird.orders, lwd = 2)
for (i in 1:10) ltt.lines(TR[[i]], lty = 2)
legend(-10, 5, lwd = c(2, 1), lty = c(1, 2), bty = "n",
       legend = c("Bird orders", "Random trees"))
```

mantel.test *Mantel Test for Similarity of Two Matrices*

Description

This function computes Mantel's permutation test for similarity of two matrices. It permutes the rows and columns of the two matrices randomly and calculates a Z-statistic.

Usage

```
mantel.test(m1, m2, nperm = 1000, graph = FALSE, ...)
```

Arguments

`m1`	a numeric matrix giving a measure of pairwise distances, correlations, or similarities among observations.
`m2`	a second numeric matrix giving another measure of pairwise distances, correlations, or similarities among observations.
`nperm`	the number of times to permute the data.
`graph`	a logical indicating whether to produce a summary graph (by default the graph is not plotted).
`...`	further arguments to be passed to `plot()` (to add a title, change the axis labels, and so on).

Details

The function calculates a Z-statistic for the Mantel test, equal to the sum of the pairwise product of the lower triangles of the permuted matrices, for each permutation of rows and columns. It compares the permuted distribution with the Z-statistic observed for the actual data.

If `graph = TRUE`, the functions plots the density estimate of the permutation distribution along with the observed Z-statistic as a vertical line.

The `...` argument allows the user to give further options to the `plot` function: the title main be changed with `main=`, the axis labels with `xlab =`, and `ylab =`, and so on.

Value

`z.stat`	the Z-statistic (sum of rows*columns of lower triangle) of the data matrices.
`p`	P-value (quantile of the observed Z-statistic in the permutation distribution).

Author(s)

Original code in S by Ben Bolker ⟨bolker@zoo.ufl.edu⟩, ported to R by Julien Claude ⟨claude@isem.univ-montp2.fr⟩

References

Mantel, N. (1967) The detection of disease clustering and a generalized regression approach. *Cancer Research*, **27**, 209–220.

Manly, B. F. J. (1986) *Multivariate statistical methods: a primer.* London: Chapman & Hall.

Examples

```
q1 <- matrix(runif(36), nrow = 6)
q2 <- matrix(runif(36), nrow = 6)
mantel.test(q1, q2, graph = TRUE,
            main = "Mantel test: a random example with 6 X 6 matrices",
            xlab = "z-statistic", ylab = "Density",
            sub = "The vertical line shows the observed z-statistic")
```

`matexpo` *Matrix Exponential*

Description

This function computes the exponential of a square matrix using a spectral decomposition.

Usage

```
matexpo(x)
```

Arguments

`x`	a square matrix of mode numeric.

Value

a numeric matrix of the same dimensions than ‘x’.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

Examples

```
### a simple rate matrix:
m <- matrix(0.1, 4, 4)
diag(m) <- -0.3
### towards equilibrium:
for (t in c(1, 5, 10, 50)) print(matexpo(m*t))
```

mcmc.popsize *Reversible Jump MCMC to Infer Demographic History*

Description

These functions implement a reversible jump MCMC framework to infer the demographic history, as well as corresponding confidence bands, from a genealogical tree. The computed demographic history is a continous and smooth function in time. `mcmc.popsize` runs the actual MCMC chain and outputs information about the sampling steps, `extract.popsize` generates from this MCMC output a table of population size in time, and `plot.popsize` and `lines.popsize` provide utility functions to plot the corresponding demographic functions.

Usage

```
mcmc.popsize(tree,nstep, thinning=1, burn.in=0,progress.bar=TRUE,
    method.prior.changepoints=c("hierarchical", "fixed.lambda"), max.nodes=30,
   lambda=0.5, gamma.shape=0.5, gamma.scale=2,
    method.prior.heights=c("skyline", "constant", "custom"),
    prior.height.mean,
    prior.height.var)
extract.popsize(mcmc.out, credible.interval=0.95, time.points=200, thinning=1, burn
## S3 method for class 'popsize':
plot(x, show.median=TRUE, show.years=FALSE, subst.rate, present.year, ...)
## S3 method for class 'popsize':
lines(x, show.median=TRUE,show.years=FALSE, subst.rate, present.year, ...)
```

Arguments

`tree`	Either an ultrametric tree (i.e. an object of class `"phylo"`), or coalescent intervals (i.e. an object of class `"coalescentIntervals"`).
`nstep`	Number of MCMC steps, i.e. length of the Markov chain (suggested value: 10,000-50,000).
`thinning`	Thinning factor (suggest value: 10-100).
`burn.in`	Number of steps dropped from the chain to allow for a burn-in phase (suggest value: 1000).
`progress.bar`	Show progress bar during the MCMC run.
`method.prior.changepoints`	If `hierarchical`is chosen (the default) then the smoothing parameter lambda is drawn from a gamma distribution with some specified shape and scale parameters. Alternatively, for `fixed.lambda` the value of lambda is a given constant.
`max.nodes`	Upper limit for the number of internal nodes of the approximating spline (default: 30).

`lambda`	Smoothing parameter. For `method="fixed.lambda"` the specifed value of lambda determines the mean of the prior distribution for the number of internal nodes of the approximating spline for the demographic function (suggested value: 0.1-1.0).
`gamma.shape`	Shape parameter of the gamma function from which `lambda` is drawn for `method="hierarchical"`.
`gamma.scale`	Scale parameter of the gamma function from which `lambda` is drawn for `method="hierarchical"`.
`method.prior.heights`	Determines the prior for the heights of the change points. If `custom` is chosen then two functions describing the mean and variance of the heigths in depence of time have to be specified (via `prior.height.mean` and `prior.height.var` options). Alternatively, two built-in priors are available: `constant` assumes constant population size and variance determined by Felsenstein (1992), and `skyline` assumes a skyline plot (see Opgen-Rhein et al. 2004 for more details).
`prior.height.mean`	Function describing the mean of the prior distribution for the heights (only used if `method.prior.heights = custom`).
`prior.height.var`	Function describing the variance of the prior distribution for the heights (only used if `method.prior.heights = custom`).
`mcmc.out`	Output from `mcmc.popsize` - this is needed as input for `extract.popsize`.
`credible.interval`	Probability mass of the confidence band (default: 0.95).
`time.points`	Number of discrete time points in the table output by `extract.popsize`.
`x`	Table with population size versus time, as computed by `extract.popsize`.
`show.median`	Plot median rather than mean as point estimate for demographic function (default: TRUE).
`show.years`	Option that determines whether the time is plotted in units of of substitutions (default) or in years (requires specification of substution rate and year of present).
`subst.rate`	Substitution rate (see option show.years).
`present.year`	Present year (see option show.years).
`...`	Further arguments to be passed on to `plot`.

Details

Please refer to Opgen-Rhein et al. (2004) for methodological details, and the help page of `skyline` for information on a related approach.

Author(s)

Rainer Opgen-Rhein (`http://www.stat.uni-muenchen.de/~opgen/`) and Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`). Parts of the rjMCMC sampling procedure are adapted from R code by Karl Browman (`http://www.biostat.jhsph.edu/~kbroman/`)

References

Opgen-Rhein, R., Fahrmeir, L. and Strimmer, K. 2005. Inference of demographic history from genealogical trees using reversible jump Markov chain Monte Carlo. *BMC Evolutionary Biology*, **5**, 6.

See Also

skyline and skylineplot.

Examples

```
# get tree
data("hivtree.newick") # example tree in NH format
tree.hiv <- read.tree(text = hivtree.newick) # load tree

# run mcmc chain
mcmc.out <- mcmc.popsize(tree.hiv, nstep=100, thinning=1, burn.in=0,progress.bar=FALSE) # tc
#mcmc.out <- mcmc.popsize(tree.hiv, nstep=10000, thinning=5, burn.in=500) # remove comments!

# make list of population size versus time
popsize  <- extract.popsize(mcmc.out)

# plot and compare with skyline plot
sk <- skyline(tree.hiv)
plot(sk, lwd=1, lty=3, show.years=TRUE, subst.rate=0.0023, present.year = 1997)
lines(popsize, show.years=TRUE, subst.rate=0.0023, present.year = 1997)
```

---

mlphylo *Estimating Phylogenies by Maximum Likelihood*

Description

mlphylo estimates a phylogenetic tree by maximum likelihood given a set of DNA sequences. The model of evolution is specified with the function DNAmodel.

logLik, deviance, and AIC are generic functions used to extract the log-likelihood, the deviance (-2*logLik), or the Akaike information criterion of a tree. If no such values are available, NULL is returned.

Usage

```
mlphylo(x, phy, model = DNAmodel(), search.tree = FALSE,
        quiet = FALSE, value = NULL, fixed = FALSE)
## S3 method for class 'phylo':
logLik(object, ...)
## S3 method for class 'phylo':
deviance(object, ...)
## S3 method for class 'phylo':
AIC(object, ..., k = 2)
```

Arguments

`x`	an object of class `"DNAbin"` giving the (aligned) DNA sequence data.
`phy`	an object of class `"phylo"` giving the tree.
`model`	an object of class `"DNAmodel"` giving the model to be fitted.
`search.tree`	a logical specifying whether to search for the best tree (defaults to FALSE) (not functional for the moment).
`quiet`	a logical specifying whether to display the progress of the analysis.
`value`	a list with elements named `rates`, `alpha`, and `invar`, or at least one of these, giving the initial values of the parameters of the model. If `NULL`, some initial values are given internally.
`fixed`	a logical specifying whether to optimize parameters given in `value`.
`object`	an object of class `"phylo"`.
`k`	a numeric value giving the penalty per estimated parameter; the default is `k = 2` which is the classical Akaike information criterion.
`...`	further arguments passed to or from other methods.

Details

The model specified by `DNAmodel` is fitted using the standard "pruning" algorithm of Felsenstein (1981).

The implementation of the inter-sites variation in substitution rates follows the methodology developed by Yang (1994).

The difference among partitions is parametrized with a contrast parameter (denoted ξ) that specifies the contrast in mean susbtitution rate among the partitions. This methodology is inspired from one introduced by Yang (1996).

The substitution rates are indexed column-wise in the rate matrix: the first rate is set to one.

Value

an object of class `"phylo"`. There are possible additional attributes:

`loglik`	the maximum log-likelihood.
`npart`	the number of partitions.
`model`	the substitution model.
`rates`	the estimated substitution rates.
`invar`	the estimated proportion of invariants.
`alpha`	the estimated shape parameter of the inter-sites variation in substitution rates.

Note

For the moment, it is not possible to estimate neither branch lengths, nor the topology with `mlphylo`. The function may estimate all other parameters: substitution rates, shape (α) of the inter-sites variation in substitution rates, the proportion of invariants, and the "contrast" parameter (ξ) among partitions.

Alternative topologies can also be compared using likelihood-ratio tests (LRTs) or AICs.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17**, 368–376.

Yang, Z. (1994) Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. *Journal of Molecular Evolution*, **39**, 306–314.

Yang, Z. (1996) Maximum-likelihood models for combined analyses of multiple sequence data. *Journal of Molecular Evolution*, **42**, 587–596.

See Also

`DNAmodel`, `nj`, `read.dna`, `summary.phylo`, `bionj`, `fastme`

`mrca` *Find Most Recent Common Ancestors Between Pairs*

Description

This function returns for each pair of tips (and nodes) its most recent common ancestor (MRCA).

Usage

```
mrca(phy, full = FALSE)
```

Arguments

`phy`	an object of class `"phylo"`.
`full`	a logical indicating whether to return the MRCAs among all tips and nodes (if `TRUE`); the default is to return only the MRCAs among tips.

Details

The diagonal is set to the number of the tips (and nodes if `full = TRUE`).

If `full = FALSE`, the colnames and rownames are set with the tip labels of the tree; otherwise the numbers are given as names.

Value

a matrix of mode numeric.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

`mst` *Minimum Spanning Tree*

Description

The function `mst` finds the minimum spanning tree between a set of observations using a matrix of pairwise distances.

The `plot` method plots the minimum spanning tree showing the links where the observations are identified by their numbers.

Usage

```
mst(X)
## S3 method for class 'mst':
plot(x, graph = "circle", x1 = NULL, x2 = NULL, ...)
```

Arguments

`X`	either a matrix that can be interpreted as a distance matrix, or an object of class `"dist"`.
`x`	an object of class `"mst"` (e.g. returned by `mst()`).
`graph`	a character string indicating the type of graph to plot the minimum spanning tree; two choices are possible: `"circle"` where the observations are plotted regularly spaced on a circle, and `"nsca"` where the two first axes of a non-symmetric correspondence analysis are used to plot the observations (see Details below). If both arguments `x1` and `x2` are given, the argument `graph` is ignored.
`x1`	a numeric vector giving the coordinates of the observations on the *x*-axis. Both `x1` and `x2` must be specified to be used.
`x2`	a numeric vector giving the coordinates of the observations on the *y*-axis. Both `x1` and `x2` must be specified to be used.
`...`	further arguments to be passed to `plot()`.

Details

These functions provide two ways to plot the minimum spanning tree which try to space as much as possible the observations in order to show as clearly as possible the links. The option `graph = "circle"` simply plots regularly the observations on a circle, whereas `graph = "nsca"` uses a non-symmetric correspondence analysis where each observation is represented at the centroid of its neighbours.

Alternatively, the user may use any system of coordinates for the obsevations, for instance a principal components analysis (PCA) if the distances were computed from an original matrix of continous variables.

Value

an object of class `"mst"` which is a square numeric matrix of size equal to the number of observations with either `1` if a link between the corresponding observations was found, or `0` otherwise. The names of the rows and columns of the distance matrix, if available, are given as rownames and colnames to the returned object.

Author(s)

Yvonnick Noel ⟨noel@univ-lille3.fr⟩, Julien Claude ⟨claude@isem.univ-montp2.fr⟩ and Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`dist.dna`, `dist.gene`, `dist`, `plot`

Examples

```
library(stats)
n <- 20
X <- matrix(runif(n * 10), n, 10)
d <- dist(X)
PC <- prcomp(X)
M <- mst(d)
opar <- par()
par(mfcol = c(2, 2))
plot(M)
plot(M, graph = "nsca")
plot(M, x1 = PC$x[, 1], x2 = PC$x[, 2])
par(opar)
```

`multi2di` *Collapse and Resolve Multichotomies*

Description

These two functions collapse or resolve multichotomies in phylogenetic trees.

Usage

```
multi2di(phy, random = TRUE)
di2multi(phy, tol = 1e-08)
```

Arguments

`phy`	an object of class `"phylo"`.
`random`	a logical value specifying whether to resolve the multichotomies randomly (the default) or in the order they appear in the tree (if `random = FALSE`).
`tol`	a numeric value giving the tolerance to consider a branch length significantly greater than zero.

Details

`multi2di` transforms all multichotomies into a series of dichotomies with one (or several) branch(es) of length zero.

`di2multi` deletes all branches smaller than `tol` and collapses the corresponding dichotomies into a multichotomy.

Value

Both functions return an object of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`is.binary.tree`

Examples

```
data(bird.families)
is.binary.tree(bird.families)
is.binary.tree(multi2di(bird.families))
all.equal(di2multi(multi2di(bird.families)), bird.families)
### To see the results of randomly resolving a trichotomy:
tr <- read.tree(text = "(a:1,b:1,c:1);")
layout(matrix(1:4, 2, 2))
for (i in 1:4)
  plot(multi2di(tr), use.edge.length = FALSE, cex = 1.5)
layout(matrix(1))
```

`nj` *Neighbor-Joining Tree Estimation*

Description

This function performs the neighbor-joining tree estimation of Saitou and Nei (1987).

Usage

```
nj(X)
```

Arguments

`X`	a distance matrix; may be an object of class “dist”.

Value

an object of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, **4**, 406–425.

See Also

`write.tree`, `read.tree`, `dist.dna`, `mlphylo`, `bionj`, `fastme`

Examples

```
### From Saitou and Nei (1987, Table 1):
x <- c(7, 8, 11, 13, 16, 13, 17, 5, 8, 10, 13,
       10, 14, 5, 7, 10, 7, 11, 8, 11, 8, 12,
       5, 6, 10, 9, 13, 8)
M <- matrix(0, 8, 8)
M[row(M) > col(M)] <- x
M[row(M) < col(M)] <- x
rownames(M) <- colnames(M) <- 1:8
tr <- nj(M)
plot(tr, "u")
### a less theoretical example
data(woodmouse)
trw <- nj(dist.dna(woodmouse))
plot(trw)
```

`node.depth` *Depth of Nodes and Tips*

Description

This function returns the depth of nodes and tips given by the number of descendants (1 is returned for tips).

Usage

```
node.depth(phy)
```

Arguments

`phy`	an object of class "phylo".

Details

The depth of a node is computed as the number of tips which are its descendants. The value of 1 is given to the tips.

Value

A numeric vector indexed with the node numbers of the matrix 'edge' of `phy`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`plot.phylo`

`nodelabels` *Labelling the Nodes, Tips, and Edges of a Tree*

Description

These functions add labels to or near the nodes, the tips, or the edges of a tree using text or plotting symbols. The text can be framed.

Usage

```
nodelabels(text, node, adj = c(0.5, 0.5), frame = "rect",
           pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
           col = "black", bg = "lightblue", ...)
tiplabels(text, tip, adj = c(0.5, 0.5), frame = "rect",
          pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
          col = "black", bg = "yellow", ...)
edgelabels(text, edge, adj = c(0.5, 0.5), frame = "rect",
           pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
           col = "black", bg = "lightgreen", ...)
```

Arguments

`text`	a vector of mode character giving the text to be printed. Can be left empty.
`node`	a vector of mode numeric giving the numbers of the nodes where the text or the symbols are to be printed. Can be left empty.
`tip`	a vector of mode numeric giving the numbers of the tips where the text or the symbols are to be printed. Can be left empty.
`edge`	a vector of mode numeric giving the numbers of the edges where the text or the symbols are to be printed. Can be left empty.
`adj`	one or two numeric values specifying the horizontal and vertical, respectively, justification of the text. By default, the text is centered horizontally and vertically. If a single value is given, this alters only the horizontal position of the text.

`frame`	a character string specifying the kind of frame to be printed around the text. This must be one of "rect" (the default), "circle", "none", or any unambiguous abbreviation of these.
`pch`	a numeric giving the type of plotting symbol to be used; this is eventually recycled. See `par` for R's plotting symbols. If `pch` is used, then `text` is ignored.
`thermo`	a numeric vector giving some proportions (values between 0 and 1) for each node, or a numeric matrix giving some proportions (the rows must sum to one).
`pie`	same than `thermo`.
`piecol`	a list of colours (given as a character vector) to be used by `thermo` or `pie`; if left `NULL`, a series of colours given by the function `rainbow` is used.
`col`	a character string giving the color to be used for the text or the plotting symbols; this is eventually recycled.
`bg`	a character string giving the color to be used for the background of the text frames or of the plotting symbols if it applies; this is eventually recycled.
`...`	further arguments passed to the `text` or `points` functions (e.g. `cex` to alter the size of the text or the symbols, or `font` for the text; see the examples below).

Details

These three functions have the same optional arguments and the same functioning.

If the arguments `text` is missing and `pch` and `thermo` are left as `NULL`, then the numbers of the nodes (or of the tips) are printed.

If `node`, `tip`, or `edge` is missing, then the text or the symbols are printed on all nodes, tips, or edges.

The option `cex` can be used to change the size of all types of labels.

A simple call of these functions with no arguments (e.g., `nodelabels()`) prints the numbers of all nodes (or tips).

In the case of `tiplabels`, it would be useful to play with the options `x.lim` and `label.offset` (and possibly `show.tip.label`) of `plot.phylo` in most cases (see the examples).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩, Ben Bolker ⟨bolker@zoo.ufl.edu⟩, and Jim Lemon

See Also

`plot.phylo`

Examples

```
tr <- read.tree(text = "((Homo,Pan),Gorilla);")
plot(tr)
nodelabels("7.3 Ma", 4, frame = "r", bg = "yellow", adj = 0)
nodelabels("5.4 Ma", 5, frame = "c", bg = "tomato", font = 3)
```

```
data(bird.orders)
plot(bird.orders, use.edge.length = FALSE, font = 1)
bs <- round(runif(22, 90, 100), 0) # some imaginary bootstrap values
bs2 <- round(runif(22, 90, 100), 0)
bs3 <- round(runif(22, 90, 100), 0)
nodelabels(bs, adj = 1.2)
nodelabels(bs2, adj = -0.2, bg = "yellow")

### something more classical
plot(bird.orders, use.edge.length = FALSE, font = 1)
nodelabels(bs, adj = -0.2, frame = "n", cex = 0.8)
nodelabels(bs2, adj = c(1.2, 1), frame = "n", cex = 0.8)
nodelabels(bs3, adj = c(1.2, -0.2), frame = "n", cex = 0.8)

### the same but we play with the font
plot(bird.orders, use.edge.length = FALSE, font = 1)
nodelabels(bs, adj = -0.2, frame = "n", cex = 0.8, font = 2)
nodelabels(bs2, adj = c(1.2, 1), frame = "n", cex = 0.8, font = 3)
nodelabels(bs3, adj = c(1.2, -0.2), frame = "n", cex = 0.8)

plot(bird.orders, "c", use.edge.length = FALSE, font = 1)
nodelabels(thermo = runif(22), cex = .8)

plot(bird.orders, "u", FALSE, font = 1, lab4ut = "a")
nodelabels(cex = .75, bg = "yellow")

### representing two characters at the tips (you could have as many
### as you want)
plot(bird.orders, "c", FALSE, font = 1, label.offset = 3,
     x.lim = 31, no.margin = TRUE)
tiplabels(pch = 21, bg = gray(1:23/23), cex = 2, adj = 1.4)
tiplabels(pch = 19, col = c("yellow", "red", "blue"), adj = 2.5, cex = 2)
### This can be used to highlight tip labels:
plot(bird.orders, font = 1)
i <- c(1, 7, 18)
tiplabels(bird.orders$tip.label[i], i, adj = 0)
### Some random data to compare piecharts and thermometres:
tr <- rtree(15)
x <- runif(14, 0, 0.33)
y <- runif(14, 0, 0.33)
z <- runif(14, 0, 0.33)
x <- cbind(x, y, z, 1 - x - y - z)
layout(matrix(1:2, 1, 2))
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(pie = x, cex = 1.3)
text(4.5, 15, "Are you \"pie\"...", font = 4, cex = 1.5)
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(thermo = x, col = rainbow(4), cex = 1.3)
text(4.5, 15, "... or \"thermo\"?", font = 4, cex = 1.5)
layout(matrix(1))
plot(tr, main = "Showing Edge Lengths")
edgelabels(round(tr$edge.length, 3), srt = 90)
plot(tr, "p", FALSE)
```

```
edgelabels("above", adj = c(0.5, -0.25), bg = "yellow")
edgelabels("below", adj = c(0.5, 1.25), bg = "lightblue")
```

`nuc.div` *Nucleotide Diversity*

Description

This function computes the nucleotide diversity from a sample of DNA sequences.

Usage

```
nuc.div(x, variance = FALSE, pairwise.deletion = FALSE)
```

Arguments

`x`	a matrix or a list which contains the DNA sequences.
`variance`	a logical indicating whether to compute the variance of the estimated nucleotide diversity.
`pairwise.deletion`	a logical indicating whether to delete the sites with missing data in a pairwise way. The default is to delete the sites with at least one missing data for all sequences.

Details

The nucleotide diversity is the sum of the number of differences between pairs of sequences divided by the number of comparisons (i.e. n(n - 1)/2, where n is the number of sequences).

The variance of the estimated diversity uses formula (10.9) from Nei (1987). This applies only if all sequences are of the same lengths, and cannot be used if `pairwise.deletion = TRUE`. A bootstrap estimate may be in order if you insist on using the latter option.

Value

A numeric vector with one or two values (if `variance = TRUE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Nei, M. (1987) *Molecular evolutionary genetics*. New York: Columbia University Press.

See Also

`base.freq`, `GC.content`, `theta.s`, `seg.sites`

Examples

```
data(woodmouse)
nuc.div(woodmouse)
nuc.div(woodmouse, TRUE)
nuc.div(woodmouse, FALSE, TRUE)
```

---

opsin *Gene Tree of 32 opsin Sequences*

Description

This data set describes a gene tree estimated from 32 opsin sequences.

Usage

```
data(opsin.newick)
```

Format

`opsin.newick` is a string with the tree in Newick format.

Source

This tree is described in Misawa and Tajima (2000) as an example for application of the Klastorin (1982) classification method.

References

Misawa, K. (2000) A simple method for classifying genes and a bootstrap test for classifications. *Molecular Biology and Evolution*, **17**, 1879–1884.

See Also

`klastorin`.

Examples

```
# example tree in NH format (a string)
data("opsin.newick")
opsin.newick

# get corresponding phylo object
tree.opsin <- read.tree(text = opsin.newick)

# plot tree
plot(tree.opsin, label.offset = 0.01)
```

phymltest *Fits a Bunch of Models with PHYML*

Description

This function calls the software PHYML and fits successively 28 models of DNA evolution. The results are saved on disk, as PHYML usually does, and returned in R as a vector with the log-likelihood value of each model.

Usage

```
phymltest(seqfile, format = "interleaved", itree = NULL,
          exclude = NULL, execname, path2exec = NULL)
## S3 method for class 'phymltest':
print(x, ...)
## S3 method for class 'phymltest':
summary(object, ...)
## S3 method for class 'phymltest':
plot(x, main = NULL, col = "blue", ...)
```

Arguments

`seqfile`	a character string giving the name of the file that contains the DNA sequences to be analysed by PHYML.
`format`	a character string specifying the format of the DNA sequences: either `"interleaved"` (the default), or `"sequential"`.
`itree`	a character string giving the name of a file with a tree in Newick format to be used as an initial tree by PHYML. If `NULL` (the default), PHYML uses a “BIONJ” tree.
`exclude`	a vector of mode character giving the models to be excluded from the analysis. These must be among those below, and follow the same syntax.
`execname`	a character string specifying the name of the PHYML binary file. This argument can be left missing under Windows: the default name `"phyml_w32"` will then be used.
`path2exec`	a character string giving the path to the PHYML binary file. If `NULL` the file must be accessible to R (either it is in the computer path, or it is in R’s working directory).
`x`	an object of class `"phymltest"`.
`object`	an object of class `"phymltest"`.
`main`	a title for the plot; if left `NULL`, a title is made with the name of the object (use `main = ""` to have no title).
`col`	a colour used for the segments showing the AIC values (blue by default).
`...`	further arguments passed to or from other methods.

Details

The present function has been tested with version 2.4 of PHYML; it should also work with version 2.3, but it won't work with version 2.1.

Under unix-like systems, it seems necessary to run R from csh or a similar shell (sh might not work).

The user must take care to set correctly the three different paths involved here: the path to PHYML's binary, the path to the sequence file, and the path to R's working directory. The function should work if all three paths are different. Obviously, there should be no problem if they are all the same.

If the usual output files of PHYML already exist, they are not deleted and PHYML's results are appended.

The following syntax is used for the models:

"X[Y][Z]00[+I][+G]"

where "X" is the first letter of the author of the model, "Y" and "Z" are possibly other co-authors of the model, "00" is the year of the publication of the model, and "+I" and "+G" indicates whether the presence of invariant sites and/or a gamma distribution of substitution rates have been specified. Thus, Kimura's model is denoted "K80" and not "K2P". The exception to this rule is the general time-reversible model which is simple denoted "GTR" model.

The seven substitution models used are: "JC69", "K80", "F81", "F84", "HKY85", "TN93", and "GTR". These models are then altered by adding the "+I" and/or "+G", resulting thus in four variants for each of them (e.g., "JC69", "JC69+I", "JC69+G", "JC69+I+G"). Some of these models are described in the help page of `dist.dna`.

When a gamma distribution of substitution rates is specified, four categories are used (which is PHYML's default behaviour), and the “alpha” parameter is estimated from the data.

For the models with a different substition rate for transitions and transversions, these rates are left free and estimated from the data (and not constrained with a ratio of 4 as in PHYML's default).

Value

`phymltest` returns an object of class `"phymltest"`: a numeric vector with the models as names.

The `print` method prints an object of class `"phymltest"` as matrix with the name of the models, the number of free parameters, the log-likelihood value, and the value of the Akaike information criterion (AIC = -2 * loglik + 2 * number of free parameters)

The `summary` method prints all the possible likelihood ratio tests for an object of class `"phymltest"`.

The `plot` method plots the values of AIC of an object of class `"phymltest"` on a vertical scale.

Note

It is important to note that the models fitted by this function is only a small fraction of the models possible with PHYML. For instance, it is possible to vary the number of categories in the (discretized) gamma distribution of substitution rates, and many parameters can be fixed by the user. The results from the present function should rather be taken as indicative of a best model.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Posada, D. and Crandall, K. A. (2001) Selecting the best-fit model of nucleotide substitution. *Systematic Biology*, **50**, 580–601.

Guindon, S. and Gascuel, O. (2003) A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, **52**, 696–704. http://atgc.lirmm.fr/phyml/

See Also

`read.tree`, `write.tree`, `dist.dna`

Examples

```
### A `fake' example with random likelihood values: it does not
### make sense, but does not need PHYML and gives you a flavour
### of what the output looks like:
x <- runif(28, -100, -50)
names(x) <- .phymltest.model
class(x) <- "phymltest"
x
summary(x)
plot(x)
plot(x, main = "", col = "red")
### This example needs PHYML, copy/paste or type the
### following commands if you want to try them, eventually
### changing setwd() and the options of phymltest()
## Not run:
setwd("D:/phyml_v2.4/exe") # under Windows
data(woodmouse)
write.dna(woodmouse, "woodmouse.txt")
X <- phymltest("woodmouse.txt")
X
summary(X)
plot(X)
## End(Not run)
```

`pic` *Phylogenetically Independent Contrasts*

Description

Compute the phylogenetically independent contrasts using the method described by Felsenstein (1985).

Usage

```
pic(x, phy, scaled = TRUE, var.contrasts = FALSE)
```

Arguments

`x`	a numeric vector.
`phy`	an object of class `"phylo"`.
`scaled`	logical, indicates whether the contrasts should be scaled with their expected variance (default to `TRUE`).
`var.contrasts`	logical, indicates whether the expected variance of the contrasts should be returned (default to `FALSE`).

Details

If `x` has names, its values are matched to the tip labels of `phy`, otherwise its values are taken to be in the same order than the tip labels of `phy`.

The user must be careful here since the function requires that both series of names perfectly match, so this operation may fail if there is a typing or syntax error. If both series of names do not match, the values in the `x` are taken to be in the same order than the tip labels of `phy`, and a warning message is issued.

Value

either a vector of phylogenetically independent contrasts (if `var.contrasts = FALSE`), or a two-column matrix with the phylogenetically independent contrasts in the first column and their expected variance in the second column (if `var.contrasts = TRUE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Felsenstein, J. (1985) Phylogenies and the comparative method. *American Naturalist*, **125**, 1–15.

See Also

`read.tree`, `compar.gee`, `compar.lynch`

Examples

```
### The example in Phylip 3.5c (originally from Lynch 1991)
cat("((((Homo:0.21,Pongo:0.21):0.28,",
   "Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);",
   file = "ex.tre", sep = "\n")
tree.primates <- read.tree("ex.tre")
X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
```

```
names(X) <- names(Y) <- c("Homo", "Pongo", "Macaca", "Ateles", "Galago")
pic.X <- pic(X, tree.primates)
pic.Y <- pic(Y, tree.primates)
cor.test(pic.X, pic.Y)
lm(pic.Y ~ pic.X - 1) # both regressions
lm(pic.X ~ pic.Y - 1) # through the origin
unlink("ex.tre") # delete the file "ex.tre"
```

`plot.ancestral` *Plot Ancestral Character Values on a Tree*

Description

Plot a phylogenetic tree with edge colors picked according to the corresponding node ancestral character value.

Usage

```
## S3 method for class 'ancestral':
plot(x, which = names(x$node.character), n.col = 10,
col.fun = function(n) rainbow(n, start = 0.4, end = 0),
plot.node.values = FALSE,
ask = prod(par("mfcol")) < length(which) && dev.interactive(),
...)
```

Arguments

`x`	An object of class 'ancestral'.
`which`	Which characters to plot. Can be a vecotr of names, or a vector of indices.
`n.col`	The number of colors to use in the gradient.
`col.fun`	the color function to use.
`plot.node.values`	Should character values used as node labels?
`ask`	Ask before each plot?
`...`	Further parameters to pass to the plot.phylo function.

Details

This function produces one plot by selected ancestral character. It uses the plot.phylo function with particular arguments to display edge colors according to ancestral character values.

Author(s)

Julien Dutheil ⟨Julien.Dutheil@univ-montp2.fr⟩

See Also

plot.phylo, evolve.phylo

Examples

```
data(bird.orders)
x <- rep(0, 4)
names(x) <- c("A", "B", "C", "D")
anc <- evolve.phylo(bird.orders, x, 1)
plot(anc, edge.width = 3, plot.node.values = TRUE)
par(mfrow = c(2, 2), mar = c(5.5, 0, 0, 0))
plot(anc, edge.width = 3, type = "r")
```

`plot.correlogram` *Plot a Correlogram*

Description

These functions plot correlagrams previously computed with `correlogram.formula`.

Usage

```
## S3 method for class 'correlogram':
plot(x, legend = TRUE, test.level = 0.05,
                col = c("grey", "red"), type = "b", xlab = "",
                ylab = "Moran's I", pch = 21, cex = 2, ...)
## S3 method for class 'correlogramList':
plot(x, lattice = TRUE, legend = TRUE,
                test.level = 0.05, col = c("grey", "red"),
                xlab = "", ylab = "Moran's I",
                type = "b", pch = 21, cex = 2, ...)
```

Arguments

`x`	an object of class `"correlogram"` or of class `"correlogramList"` (both produced by `correlogram.formula`).
`legend`	should a legend be added on the plot?
`test.level`	the level used to discriminate the plotting symbols with colours considering the P-values.
`col`	two colours for the plotting symbols: the first one is used if the P-value is greater than or equal to `test.level`, the second one otherwise.
`type`	the type of plot to produce (see `plot` for possible choices).
`xlab`	an optional character string for the label on the x-axis (none by default).
`ylab`	the default label on the y-axis.
`pch`	the type of plotting symbol.

`cex`	the default size for the plotting symbols.
`lattice`	when plotting several correlograms, should they be plotted in trellis-style with lattice (the default), or together on the same plot?
`...`	other parameters passed to the `plot` or `lines` function.

Details

When plotting several correlograms with lattice, some options have no effect: `legend`, `type`, and `pch` (`pch=19` is always used in this situation).

When using `pch` between 1 and 20 (i.e., non-filled symbols, the colours specified in `col` are also used for the lines joining the points. To keep black lines, it is better to leave `pch` between 21 and 25.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`correlogram.formula`, `Moran.I`

`plot.phylo` *Plot Phylogenies*

Description

These functions plot phylogenetic trees on the current graphical device.

Usage

```
## S3 method for class 'phylo':
plot(x, type = "phylogram", use.edge.length = TRUE,
    node.pos = NULL, show.tip.label = TRUE, show.node.label = FALSE,
    edge.color = "black", edge.width = 1, font = 3,
    cex = par("cex"), adj = NULL, srt = 0, no.margin = FALSE,
    root.edge = FALSE, label.offset = 0, underscore = FALSE,
    x.lim = NULL, y.lim = NULL, direction = "rightwards",
    lab4ut = "horizontal", tip.color = "black", ...)
## S3 method for class 'multiPhylo':
plot(x, layout = 1, ...)
```

Arguments

`x`	an object of class `"phylo"` or of class `"multiPhylo"`.
`type`	a character string specifying the type of phylogeny to be drawn; it must be one of "phylogram" (the default), "cladogram", "fan", "unrooted", "radial" or any unambiguous abbreviation of these.
`use.edge.length`	a logical indicating whether to use the edge lengths of the phylogeny to draw the branches (the default) or not (if `FALSE`). This option has no effect if the object of class `"phylo"` has no 'edge.length' element.
`node.pos`	a numeric taking the value 1 or 2 which specifies the vertical position of the nodes with respect to their descendants. If `NULL` (the default), then the value is determined in relation to 'type' and 'use.edge.length' (see details).
`show.tip.label`	a logical indicating whether to show the tip labels on the phylogeny (defaults to `TRUE`, i.e. the labels are shown).
`show.node.label`	a logical indicating whether to show the node labels on the phylogeny (defaults to `FALSE`, i.e. the labels are not shown).
`edge.color`	a vector of mode character giving the colours used to draw the branches of the plotted phylogeny. These are taken to be in the same order than the component `edge` of `phy`. If fewer colours are given than the length of `edge`, then the colours are recycled.
`edge.width`	a numeric vector giving the width of the branches of the plotted phylogeny. These are taken to be in the same order than the component `edge` of `phy`. If fewer widths are given than the length of `edge`, then these are recycled.
`font`	an integer specifying the type of font for the labels: 1 (plain text), 2 (bold), 3 (italic, the default), or 4 (bold italic).
`cex`	a numeric value giving the factor scaling of the tip and node labels (Character EXpansion). The default is to take the current value from the graphical parameters.
`adj`	a numeric specifying the justification of the text strings of the labels: 0 (left-justification), 0.5 (centering), or 1 (right-justification). This option has no effect if `type = "unrooted"`. If `NULL` (the default) the value is set with respect of `direction` (see details).
`srt`	a numeric giving how much the labels are rotated in degrees (negative values are allowed resulting in clock-like rotation); the value has an effect respectively to the value of `direction` (see Examples). This option has no effect if `type = "unrooted"`.
`no.margin`	a logical. If `TRUE`, the margins are set to zero and the plot uses all the space of the device (note that this was the behaviour of `plot.phylo` up to version 0.2-1 of 'ape' with no way to modify it by the user, at least easily).
`root.edge`	a logical indicating whether to draw the root edge (defaults to FALSE); this has no effect if 'use.edge.length = FALSE' or if 'type = "unrooted"'.
`label.offset`	a numeric giving the space between the nodes and the tips of the phylogeny and their corresponding labels. This option has no effect if `type = "unrooted"`.

`underscore`	a logical specifying whether the underscores in tip labels should be written as spaces (the default) or left as are (if `TRUE`).
`x.lim`	a numeric vector of length one or two giving the limit(s) of the x-axis. If `NULL`, this is computed with respect to various parameters such as the string lengths of the labels and the branch lengths. If a single value is given, this is taken as the upper limit.
`y.lim`	same than above for the y-axis.
`direction`	a character string specifying the direction of the tree. Four values are possible: "rightwards" (the default), "leftwards", "upwards", and "downwards".
`lab4ut`	(= labels for unrooted trees) a character string specifying the display of tip labels for unrooted trees: either `"horizontal"` where all labels are horizontal (the default), or `"axial"` where the labels are displayed in the axis of the corresponding terminal branches. This option has an effect only if `type = "unrooted"`.
`tip.color`	the colours used for the tip labels, eventually recycled (see examples).
`layout`	the number of trees to be plotted simultaneously.
`...`	further arguments to be passed to `plot` or to `plot.phylo`.

Details

If `x` is a list of trees (i.e., an object of class `"multiPhylo"`), then any further argument may be passed with `...` and could be any one of those listed above for a single tree.

The font format of the labels of the nodes and the tips is the same.

If `no.margin = TRUE`, the margins are set to zero and are not restored after plotting the tree, so that the user can access the coordinates system of the plot.

The option 'node.pos' allows the user to alter the vertical position (i.e. ordinates) of the nodes. If `node.pos = 1`, then the ordinate of a node is the mean of the ordinates of its direct descendants (nodes and/or tips). If `node.pos = 2`, then the ordinate of a node is the mean of the ordinates of all the tips of which it is the ancestor. If `node.pos = NULL` (the default), then its value is determined with respect to other options: if `type = "phylogram"` then 'node.pos = 1'; if `type = "cladogram"` and `use.edge.length = FALSE` then 'node.pos = 2'; if `type = "cladogram"` and `use.edge.length = TRUE` then 'node.pos = 1'. Remember that in this last situation, the branch lengths make sense when projected on the x-axis.

If `adj` is not specified, then the value is determined with respect to `direction`: if `direction = "leftwards"` then `adj = 1` (0 otherwise).

If the arguments `x.lim` and `y.lim` are not specified by the user, they are determined roughly by the function. This may not always give a nice result: the user may check these values with the (invisibly) returned list (see "Value:").

If you resize manually the graphical device (windows or X11) you may need to replot the tree.

Value

`plot.phylo` returns invisibly a list with the following components which values are those used for the current plot:

`type`
`use.edge.length`

`node.pos`
`show.tip.label`

`show.node.label`

`font`

`cex`

`adj`

`srt`

`no.margin`

`label.offset`

`x.lim`

`y.lim`

`direction`

`tip.color`

`Ntip`

`Nnode`

Note

The argument `asp` cannot be passed with `...`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`read.tree`, `add.scale.bar`, `axisPhylo`, `nodelabels`, `plot` for the basic plotting function in R

Examples

```
### An extract from Sibley and Ahlquist (1990)
cat("(((Strix_aluco:4.2,Asio_otus:4.2):3.1,",
   "Athene_noctua:7.3):6.3,Tyto_alba:13.5);",
   file = "ex.tre", sep = "\n")
tree.owls <- read.tree("ex.tre")
plot(tree.owls)
unlink("ex.tre") # delete the file "ex.tre"

### Show the types of trees.
layout(matrix(1:6, 3, 2))
```

```
plot(tree.owls, main = "With branch lengths")
plot(tree.owls, type = "c")
plot(tree.owls, type = "u")
plot(tree.owls, use.edge.length = FALSE, main = "Without branch lengths")
plot(tree.owls, type = "c", use.edge.length = FALSE)
plot(tree.owls, type = "u", use.edge.length = FALSE)
layout(matrix(1))

data(xenarthra)
plot(xenarthra)
### remove the margins...
plot(xenarthra, no.margin = TRUE)
### ... and use a smaller font size
plot(xenarthra, no.margin = TRUE, cex = 0.8)
plot(xenarthra, type = "c", no.margin = TRUE,
     use.edge.length = FALSE, cex = 0.8)
par(mar = c(5.1, 4.1, 4.1, 2.1))

data(bird.orders)
### using random colours and thickness
plot(bird.orders,
     edge.color = sample(colors(), length(bird.orders$edge)/2),
     edge.width = sample(1:10, length(bird.orders$edge)/2, replace = TRUE))
title("Random colours and branch thickness")
### rainbow colouring...
X <- c("red", "orange", "yellow", "green", "blue", "purple")
plot(bird.orders,
     edge.color = sample(X, length(bird.orders$edge)/2, replace = TRUE),
     edge.width = sample(1:10, length(bird.orders$edge)/2, replace = TRUE))
title("Rainbow colouring")
plot(bird.orders, type = "c", use.edge.length = FALSE,
     edge.color = sample(X, length(bird.orders$edge)/2, replace = TRUE),
     edge.width = rep(5, length(bird.orders$edge)/2))
segments(rep(0, 6), 6.5:1.5, rep(2, 6), 6.5:1.5, lwd = 5, col = X)
text(rep(2.5, 6), 6.5:1.5, paste(X, "..."), adj = 0)
title("Character mapping...")
plot(bird.orders, "u", font = 1, cex = 0.75)
data(bird.families)
plot(bird.families, "u", lab4ut = "axial", font = 1, cex = 0.5)
plot(bird.families, "r", font = 1, cex = 0.5)
### cladogram with oblique tip labels
plot(bird.orders, "c", FALSE, direction = "u", srt = -40, x.lim = 25.5)
### facing trees with different informations...
tr <- bird.orders
tr$tip.label <- rep("", 23)
layout(matrix(1:2, 1, 2), c(5, 4))
plot(bird.orders, "c", FALSE, adj = 0.5, no.margin = TRUE, label.offset = 0.8,
     edge.color = sample(X, length(bird.orders$edge)/2, replace = TRUE),
     edge.width = rep(5, length(bird.orders$edge)/2))
text(7.5, 23, "Facing trees with\ndifferent informations", font = 2)
plot(tr, "p", direction = "l", no.margin = TRUE,
     edge.width = sample(1:10, length(bird.orders$edge)/2, replace = TRUE))
### Recycling of arguments gives a lot of possibilities
### for tip labels:
plot(bird.orders, tip.col = c(rep("red", 5), rep("blue", 18)),
     font = c(rep(3, 5), rep(2, 17), 1))
plot(bird.orders, tip.col = c("blue", "green"),
     cex = 23:1/23 + .3, font = 1:3)
co <- c(rep("blue", 9), rep("green", 35))
plot(bird.orders, "f", edge.col = co)
plot(bird.orders, edge.col = co)
layout(1)
```

`plot.varcomp` *Plot Variance Components*

Description

Plot previously estimated variance components.

Usage

```
## S3 method for class 'varcomp':
plot(x, xlab = "Levels", ylab = "Variance", type = "b", ...)
```

Arguments

`x`	A *varcomp* object
`xlab`	x axis label
`ylab`	y axis label
`type`	plot type ("l", "p" or "b", see `plot`)
`...`	Further argument sent to the `xyplot` function.

Value

The same as `xyplot`.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

See Also

`varcomp`

print.phylo *Compact Display of a Phylogeny*

Description

These functions prints a compact summary of a phylogeny, or a list of, on the console.

The operators `[`, `[[`, and `$` propagate the class correctly.

Usage

```
## S3 method for class 'phylo':
print(x, printlen = 6 ,...)
## S3 method for class 'multiPhylo':
print(x, details = FALSE ,...)
## S3 method for class 'multiPhylo':
x[i]
## S3 method for class 'multiPhylo':
x[[i]]
## S3 method for class 'multiPhylo':
x$name
## S3 method for class 'multiPhylo':
str(object, ...)
```

Arguments

`x`	an object of class `"phylo"` or `"multiPhylo"`.
`object`	an object of class `"multiPhylo"`.
`printlen`	the number of labels to print (6 by default).
`details`	a logical indicating whether to print information on all trees.
`i`	indices of the tree(s) to select from a list; this may be a vector of integers, logicals, or names.
`name`	a character string specifying the tree to be extracted.
`...`	further arguments passed to or from other methods.

Value

An object of class `"phylo"` (`[[`, `$`) or of class `"multiPhylo"` (`[[`), or NULL.

Author(s)

Ben Bolker ⟨bolker@zoo.ufl.edu⟩ and Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`read.tree`, `summary.phylo`, `print` for the generic R function

`ratogram` *Ratogram Computed by Nonparametric Rate Smoothing*

Description

`ratogram` computes a ratogram from a phylogram by applying the NPRS (nonparametric rate smoothing) algorithm described in Sanderson (1997).

Usage

```
ratogram(phy, scale = 1, expo = 2, minEdgeLength = 1e-06)
```

Arguments

`phy`	A phylogenetic tree (i.e. an object of class `"phylo"`), where the branch lengths are measured in substitutions.
`scale`	Age of the root in the chronogram corresponding to the inferred ratogram(default value: 0).
`expo`	Exponent in the objective function (default value: 2)
`minEdgeLength`	Minimum edge length in the phylogram (default value: 1e-06). If any branch lengths are smaller then they will be set to this value.

Details

Please refer to Sanderson (1997) for mathematical details

Value

`chronogram` returns an object of class `"phylo"`. The branch lengths of this tree will be the absolute rates estimated for each branch.

Author(s)

Gangolf Jobb (`http://www.treefinder.de`) and Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

References

Sanderson, M. J. (1997) A nonparametric approach to estimating divergence times in the absence of rate constancy. *Molecular Biology and Evolution*, **14**, 1218–1231.

See Also

`chronogram`, `NPRS.criterion`.

Examples

```
# get tree
data("landplants.newick") # example tree in NH format
tree.landplants <- read.tree(text = landplants.newick)

# plot tree
tree.landplants
plot(tree.landplants, label.offset = 0.001)

# estimate ratogram
rato.plants <- ratogram(tree.landplants)

# plot
plot(rato.plants, label.offset = 0.001)
```

`read.GenBank` *Read DNA Sequences from GenBank via Internet*

Description

This function connects to the GenBank database, and reads nucleotide sequences using accession numbers given as arguments.

Usage

```
read.GenBank(access.nb, seq.names = access.nb,
             species.names = TRUE, as.character = FALSE)
```

Arguments

`access.nb`	a vector of mode character giving the accession numbers.
`seq.names`	the names to give to each sequence; by default the accession numbers are used.
`species.names`	a logical indicating whether to attribute the species names to the returned object.
`as.character`	a logical controlling whether to return the sequences as an object of class `"DNAbin"` (the default).

Details

The function uses the site `http://www.ncbi.nlm.nih.gov/` from where the sequences are downloaded.

If `species.names = TRUE`, the returned list has an attribute `"species"` containing the names of the species taken from the field “ORGANISM” in GenBank.

Value

A list of DNA sequences made of vectors of class `"DNAbin"`, or of single characters (if `as.character = "TRUE"`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`read.dna`, `write.dna`, `dist.dna`, `DNAbin`

Examples

```
### This won't work if your computer is not connected
### to the Internet!!!
###
### Get the 8 sequences of tanagers (Ramphocelus)
### as used in Paradis (1997)
ref <- c("U15717", "U15718", "U15719", "U15720",
         "U15721", "U15722", "U15723", "U15724")
### Copy/paste or type the following commands if you
### want to try them.
## Not run:
Rampho <- read.GenBank(ref)
### get the species names:
attr(Rampho, "species")
### build a matrix with the species names and the accession numbers:
cbind(attr(Rampho, "species"), names(Rampho))
### print the first sequence
### (can be done with `Rampho$U15717' as well)
Rampho[[1]]
### print the first sequence in a cleaner way
cat(Rampho[[1]], "\n", sep = "")
## End(Not run)
```

`read.caic` *Read Tree File in CAIC Format*

Description

This function reads one tree from a CAIC file. A second file containing branch lengths values may also be passed (experimental).

Usage

```
read.caic(file, brlen = NULL, skip = 0, comment.char = "#", ...)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`brlen`	a file name for the branch lengths file.

`skip`	the number of lines of the input file to skip before beginning to read data (this is passed directly to scan()).
`comment.char`	a single character, the remaining of the line after this character is ignored (this is passed directly to scan()).
`...`	Further arguments to be passed to scan().

Details

Read a tree from a file in the format used by the CAIC and MacroCAIc program.

Value

an object of class "phylo" with the following components:

`edge`	a two-column matrix of mode character where each row represents an edge of the tree; the nodes and the tips are symbolized with numbers (these numbers are not treated as numeric, hence the mode character); the nodes are represented with negative numbers (the root being "-1"), and the tips are represented with positive numbers. For each row, the first column gives the ancestor. This representation allows an easy manipulation of the tree, particularly if it is rooted.
`edge.length`	a numeric vector giving the lengths of the branches given by edge.
`tip.label`	a vector of mode character giving the names of the tips; the order of the names in this vector corresponds to the (positive) number in edge.
`node.label`	(optional) a vector of mode character giving the names of the nodes (set to NULL if not available in the file).
`root.edge`	(optional) a numeric value giving the length of the branch at the root is it exists (NULL otherwise).

Warning

The branch length support is still experimental and was not fully tested.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Purvis, A. and Rambaut, A. (1995) Comparative analysis by independent contrasts (CAIC): an Apple Macintosh application for analysing comparative data. *CABIOS*, **11** :241–251.

See Also

`read.tree`, `read.nexus`

Examples

```
### The same example than in read.tree, without branch lengths.
### An extract from Sibley and Ahlquist (1990)
cat("AAA","Strix_aluco","AAB","Asio_otus",
   "AB","Athene_noctua","B","Tyto_alba",
   file = "ex.tre", sep = "\n")
tree.owls <- read.caic("ex.tre")
plot(tree.owls)
tree.owls
unlink("ex.tre") # delete the file "ex.tre"
```

`read.dna` *Read DNA Sequences in a File*

Description

This function reads DNA sequences in a file, and returns a matrix or a list of DNA sequences with the names of the taxa read in the file as rownames or names, respectively. By default, the sequences are stored in binary format, otherwise (if `as.character = "TRUE"`) in lower case.

Usage

```
read.dna(file, format = "interleaved", skip = 0,
         nlines = 0, comment.char = "#", seq.names = NULL,
         as.character = FALSE)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`format`	a character string specifying the format of the DNA sequences. Three choices are possible: `"interleaved"`, `"sequential"`, or `"fasta"`, or any unambiguous abbreviation of these.
`skip`	the number of lines of the input file to skip before beginning to read data.
`nlines`	the number of lines to be read (by default the file is read untill its end).
`comment.char`	a single character, the remaining of the line after this character is ignored.
`seq.names`	the names to give to each sequence; by default the names read in the file are used.
`as.character`	a logical controlling whether to return the sequences as an object of class `"DNAbin"` (the default).

Details

This function follows the interleaved and sequential formats defined in PHYLIP (Felsenstein, 1993) but with the original feature than there is no restriction on the lengths of the taxa names (though a data file with 10-characters-long taxa names is fine as well). For these two formats, the first line of the file must contain the dimensions of the data (the numbers of taxa and the numbers of nucleotides); the sequences are considered as aligned and thus must be of the same lengths for all taxa. For the FASTA format, the conventions defined in the URL below (see References) are followed; the sequences are taken as non-aligned. For all formats, the nucleotides can be arranged in any way with blanks and line-breaks inside (with the restriction that the first ten nucleotides must be contiguous for the interleaved and sequential formats, see below). The names of the sequences are read in the file unless the 'seq.names' option is used. Particularities for each format are detailed below.

Interleaved: the function starts to read the sequences when it finds 10 contiguous characters belonging to the ambiguity code of the IUPAC (namely A, C, G, T, U, M, R, W, S, Y, K, V, H, D, B, and N, upper- or lowercase, so you might run into trouble if you have a taxa name with 10 contiguous letters among these!) All characters before the sequences are taken as the taxa names after removing the leading and trailing spaces (so spaces in a taxa name are allowed). It is assumed that the taxa names are not repeated in the subsequent blocks of nucleotides.

Sequential: the same criterion than for the interleaved format is used to start reading the sequences and the taxa names; the sequences are then read until the number of nucleotides specified in the first line of the file is reached. This is repeated for each taxa.

FASTA: This looks like the sequential format but the taxa names (or rather a description of the sequence) are on separate lines beginning with a 'greater than' character ">" (there may be leading spaces before this character). These lines are taken as taxa names after removing the ">" and the possible leading and trailing spaces. All the data in the file before the first sequence is ignored.

Value

a matrix or a list (if `format = "fasta"`) of DNA sequences stored in binary format, or of mode character (if `as.character = "TRUE"`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Anonymous. FASTA format description. http://www.ncbi.nlm.nih.gov/BLAST/fasta.html

Anonymous. IUPAC ambiguity codes. http://www.ncbi.nlm.nih.gov/SNP/iupac.html

Felsenstein, J. (1993) Phylip (Phylogeny Inference Package) version 3.5c. Department of Genetics, University of Washington. http://evolution.genetics.washington.edu/phylip/phylip.html

See Also

`read.GenBank`, `write.dna`, `DNAbin`, `dist.dna`, `woodmouse`

Examples

```
### a small extract from `data(woddmouse)'
cat("3 40",
"No305     NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
"No304     ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
"No306     ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
file = "exdna.txt", sep = "\n")
ex.dna <- read.dna("exdna.txt", format = "sequential")
str(ex.dna)
ex.dna
### the same data in interleaved format...
cat("3 40",
"No305     NTTCGAAAAA CACACCCACT",
"No304     ATTCGAAAAA CACACCCACT",
"No306     ATTCGAAAAA CACACCCACT",
"          ACTAAAANTT ATCAGTCACT",
"          ACTAAAAATT ATCAACCACT",
"          ACTAAAAATT ATCAATCACT",
file = "exdna.txt", sep = "\n")
ex.dna2 <- read.dna("exdna.txt")
### ... and in FASTA format
cat("> No305",
"NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
"> No304",
"ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
"> No306",
"ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
file = "exdna.txt", sep = "\n")
ex.dna3 <- read.dna("exdna.txt", format = "fasta")
### These are the same!
identical(ex.dna, ex.dna2)
identical(ex.dna, ex.dna3)
unlink("exdna.txt") # clean-up
```

`read.nexus` *Read Tree File in Nexus Format*

Description

This function reads one or several trees in a NEXUS file.

Usage

```
read.nexus(file, tree.names = NULL)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`tree.names`	if there are several trees to be read, a vector of mode character that gives names to the individual trees; if `NULL` (the default), the trees are named `"tree1"`, `"tree2"`, ...

Details

The present implementation tries to follow as much as possible the NEXUS standard. Only the block "TREES" is read; the other data can be read with other functions (e.g., `read.dna`, `read.table`, ...). A trace of the original data is kept with the attribute `"origin"` (see below).

'read.nexus' tries to represent correctly trees with a badly represented root edge (i.e. with an extra pair of parentheses). For instance, the tree "((A:1,B:1):10);" will be read like "(A:1,B:1):10;" but a warning message will be issued in the former case as this is apparently not a valid Newick format. If there are two root edges (e.g., "(((A:1,B:1):10):10);"), then the tree is not read and an error message is issued.

Value

an object of class `"phylo"` with the following components:

`edge`	a two-column matrix of mode character where each row represents an edge of the tree; the nodes and the tips are symbolized with numbers (these numbers are not treated as numeric, hence the mode character); the nodes are represented with negative numbers (the root being `"-1"`), and the tips are represented with positive numbers. For each row, the first column gives the ancestor. This representation allows an easy manipulation of the tree, particularly if it is rooted.
`edge.length`	a numeric vector giving the lengths of the branches given by `edge`.
`tip.label`	a vector of mode character giving the names of the tips; the order of the names in this vector corresponds to the (positive) number in `edge`.
`node.label`	(optional) a vector of mode character giving the names of the nodes (set to `NULL` if not available in the file).
`root.edge`	(optional) a numeric value giving the length of the branch at the root is it exists (`NULL` otherwise).

If several trees are read in the file, the returned object is of class `"multiPhylo"`, and is a list of objects of class `"phylo"`.

An attribute `"origin"` is further given to the returned object which gives the name of the source file (with its path). This is used to write a tree in a NEXUS file where all the original data must be written (not only the tree) in accordance to the specifications of Maddison et al. (1997).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) NEXUS: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

See Also

`read.tree`, `write.nexus`, `write.tree`, `read.nexus.data`, `write.nexus.data`

`read.nexus.data` *Read Character Data In NEXUS Format*

Description

This function reads a file with sequences in the NEXUS format.

Usage

```
read.nexus.data(file)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string.

Details

This parser tries to read data from a file written in a *restricted* NEXUS format (see examples below).

Please see files ‘data.nex’ and ‘taxacharacters.nex’ for examples of formats that will work.

Some noticeable exceptions from the NEXUS standard (non-exhaustive list):

I Comments must be either on separate lines or at the end of lines. Examples:\ `[Comment]` — **OK** `Taxon ACGTACG [Comment]` — **OK** `[Comment line 1`
`Comment line 2]` — **NOT OK!** `Tax[Comment]on ACG[Comment]T` — **NOT OK!**

II No spaces (or comments) are allowed in the sequences. Examples:\ `name ACGT` — **OK** `name AC GT` — **NOT OK!**

III No spaces are allowed in taxon names, not even if names are in single quotes. That is, single-quoted names are not treated as such by the parser. Examples:\ `Genus_species` — **OK** `'Genus_species'` — **OK** `'Genus species'` — **NOT OK!**

IV The trailing `end` that closes the `matrix` must be on a separate line. Examples:\ `taxon AACCGGT`
`end;` — **OK** `taxon AACCGGT;`
`end;` — **OK** `taxon AACCCGT; end;` — **NOT OK!**

V Multistate characters are not allowed. That is, NEXUS allows you to specify multiple character states at a character position either as an uncertainty, `(XY)`, or as an actual appearance of multiple states, `{XY}`. This is information is not handled by the parser. Examples:\ `taxon 0011?110` **— OK** `taxon 0011{01}110` **— NOT OK!** `taxon 0011(01)110` **— NOT OK!**

VI The number of taxa must be on the same line as `ntax`. The same applies to `nchar`. Examples:\ `ntax = 12` **— OK** `ntax =`
`12` **— NOT OK!**

VII The word "matrix" can not occur anywhere in the file before the actual `matrix` command, unless it is in a comment. Examples:\ `BEGIN CHARACTERS;`
`TITLE 'Data in file "03a-cytochromeB.nex"';`
`DIMENSIONS NCHAR=382;`
`FORMAT DATATYPE=Protein GAP=- MISSING=?;`
`["This is The Matrix"]` **— OK**
`MATRIX\`
`BEGIN CHARACTERS;`
`TITLE 'Matrix in file "03a-cytochromeB.nex"';` **— NOT OK!**
`DIMENSIONS NCHAR=382;`
`FORMAT DATATYPE=Protein GAP=- MISSING=?;`
`MATRIX`

Value

A list of sequences each made of a single vector of mode character where each element is a (phylogenetic) character state.

Author(s)

Johan Nylander ⟨nylander@scs.fsu.edu⟩

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) NEXUS: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

See Also

`read.nexus`, `write.nexus`, `write.nexus.data`

Examples

```
## Use read.nexus.data to read a file in NEXUS format into object x
## Not run: x <- read.nexus.data("file.nex")
```

`read.tree` *Read Tree File in Parenthetic Format*

Description

This function reads a file which contains one or several trees in parenthetic format known as the Newick or New Hampshire format.

Usage

```
read.tree(file = "", text = NULL, tree.names = NULL,
          skip = 0, comment.char = "#", ...)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string; if `file = ""` (the default) then the tree is input on the keyboard, the entry being terminated with a blank line.
`text`	alternatively, the name of a variable of mode character which contains the tree(s) in parenthetic format. By default, this is ignored (set to `NULL`, meaning that the tree is read in a file); if `text` is not `NULL`, then the argument `file` is ignored.
`tree.names`	if there are several trees to be read, a vector of mode character that gives names to the individual trees; if `NULL` (the default), the trees are named `"tree1"`, `"tree2"`, ...
`skip`	the number of lines of the input file to skip before beginning to read data (this is passed directly to `scan()`).
`comment.char`	a single character, the remaining of the line after this character is ignored (this is passed directly to `scan()`).
`...`	Further arguments to be passed to `scan()`.

Details

The default option for `file` allows to type directly the tree on the keyboard (or possibly to copy from an editor and paste in R's console) with, e.g., `mytree <- read.tree()`.

'read.tree' tries to represent correctly trees with a badly represented root edge (i.e. with an extra pair of parentheses). For instance, the tree "((A:1,B:1):10);" will be read like "(A:1,B:1):10;" but a warning message will be issued in the former case as this is apparently not a valid Newick format. If there are two root edges (e.g., "(((A:1,B:1):10):10);"), then the tree is not read and an error message is issued.

Value

an object of class `"phylo"` with the following components:

`edge`	a two-column matrix of mode numeric where each row represents an edge of the tree; the nodes and the tips are symbolized with numbers; the tips are numbered 1, 2, ..., and the nodes are numbered after the tips. For each row, the first column gives the ancestor.
`edge.length`	(optional) a numeric vector giving the lengths of the branches given by `edge`.
`tip.label`	a vector of mode character giving the names of the tips; the order of the names in this vector corresponds to the (positive) number in `edge`.
`Nnode`	the number of (internal) nodes.
`node.label`	(optional) a vector of mode character giving the names of the nodes.
`root.edge`	(optional) a numeric value giving the length of the branch at the root if it exists.

If several trees are read in the file, the returned object is of class `"multiPhylo"`, and is a list of objects of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Felsenstein, J. The Newick tree format. `http://evolution.genetics.washington.edu/phylip/newicktree.html`

Olsen, G. Interpretation of the "Newick's 8:45" tree format standard. `http://evolution.genetics.washington.edu/phylip/newick_doc.html`

Paradis, E. (2006) Definition of Formats for Coding Phylogenetic Trees in R. `http://ape.mpl.ird.fr/misc/FormatTreeR_4Dec2006.pdf`

See Also

`write.tree`, `read.nexus`, `write.nexus`, `scan` for the basic R function to read data in a file

Examples

```
### An extract from Sibley and Ahlquist (1990)
cat("(((Strix_aluco:4.2,Asio_otus:4.2):3.1,",
   "Athene_noctua:7.3):6.3,Tyto_alba:13.5);",
   file = "ex.tre", sep = "\n")
tree.owls <- read.tree("ex.tre")
str(tree.owls)
tree.owls
unlink("ex.tre") # delete the file "ex.tre"
### Only the first three species using the option `text'
TREE <- "((Strix_aluco:4.2,Asio_otus:4.2):3.1,Athene_noctua:7.3);"
TREE
tree.owls.bis <- read.tree(text = TREE)
str(tree.owls.bis)
tree.owls.bis
```

`reorder.phylo` *Internal Reordering of Trees*

Description

This function changes the internal structure of a phylogeny stored as an object of class `"phylo"`. The tree returned is the same than the one input, but the ordering of the edges could be different.

Usage

```
## S3 method for class 'phylo':
reorder(x, order = "cladewise", ...)
```

Arguments

`x`	an object of class `"phylo"`.
`order`	a character string: either `"cladewise"` (the default), or `"pruningwise"`, or any unambiguous abbreviation of these.
`...`	further arguments passed to or from other methods.

Details

Because in a tree coded as an object of class `"phylo"` each branch is represented by a row in the element 'edge', there is an arbitrary choice for the ordering of these rows. `reorder` allows to reorder these rows according to two rules: in the `"cladewise"` order each clade is formed by a series of contiguous rows; this is the order returned by `read.tree`. In the `"pruningwise"` order, rows are arranged so that "pruning" the tree (or post-order tree traversal) can be done by descending along the rows of 'edge'. The possible multichotomies and branch lengths are preserved.

Value

an object of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`read.tree` to read tree files in Newick format, `reorder` for the generic function

Examples

```
data(bird.families)
tr <- reorder(bird.families, "p")
all.equal(bird.families, tr) # uses all.equal.phylo actually
all.equal.list(bird.families, tr) # bypasses the generic
```

root *Roots Phylogenetic Trees*

Description

`root` reroots a phylogenetic tree with respect to the specified outgroup or at the node specified in `node`.

`unroot` unroots a phylogenetic tree, or returns it unchanged if it is already unrooted.

`is.rooted` tests whether a tree is rooted.

Usage

```
root(phy, outgroup, node = NULL, resolve.root = FALSE)
unroot(phy)
is.rooted(phy)
```

Arguments

`phy`	an object of class `"phylo"`.
`outgroup`	a vector of mode numeric or character specifying the new outgroup.
`node`	alternatively, a node number where to root the tree.
`resolve.root`	a logical specifying whether to resolve the new root as a bifurcating node.

Details

The argument `outgroup` can be either character or numeric. In the first case, it gives the labels of the tips of the new outgroup; in the second case the numbers of these labels in the vector `phy$tip.label` are given.

If `outgroup` is of length one (i.e., a single value), then the tree is rerooted using the node below this tip as the new root.

If `outgroup` is of length two or more, the most recent common ancestor (MRCA) is used as the new root. Note that the tree is really unrooted before being rerooted, so that if `outgroup` is already the outgroup, then the returned tree is not the same than the original one (see examples). If `outgroup` is not monophyletic, the operation fails and an error message is issued.

If `resolve.root = TRUE`, `root` adds a zero-length branch below the MRCA of the ingroup.

A tree is considered rooted if either only two branches connect to the root, or if there is a `root.edge` element. In all other cases, `is.rooted` returns `FALSE`.

Value

an object of class `"phylo"` for `root` and `unroot`; a single logical value for `is.rooted`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

bind.tree, drop.tip, nodelabels, identify.phylo

Examples

```
data(bird.orders)
plot(root(bird.orders, 1))
plot(root(bird.orders, 1:5))

tr <- root(bird.orders, 1)
is.rooted(bird.orders) # yes!
is.rooted(tr)          # no!
### This is because the tree has been unrooted first before rerooting.
### You can delete the outgroup...
is.rooted(drop.tip(tr, "Struthioniformes"))
### ... or resolve the basal trichotomy in two ways:
is.rooted(multi2di(tr))
is.rooted(root(bird.orders, 1, r = TRUE))
### To keep the basal trichotomy but forcing the tree as rooted:
tr$root.edge <- 0
is.rooted(tr)
```

`rotate` *Swopping sister clades*

Description

For a given node, rotate exchanges the position of two clades descending from this node. It can handle dichotomies as well as polytomies. In the latter case, two clades from the polytomy are selected for swapping.

Usage

```
rotate(phy, node, polytom = c(1, 2))
```

Arguments

`phy`	an object of class `"phylo"`.
`node`	a vector of mode numeric or character specifying the number of the node
`polytom`	a vector of mode numeric and length two specifying the two clades that should be exchanged in a polytomy

Details

phy can be either rooted or unrooted, contain polytomies and lack branch lengths. In the presence of very short branch lengths it is convenient to plot the phylogenetic tree without branch lengths in order to identify the number of the node in question.

node can be any of the interior nodes of a phylogenetic tree including the root node. Number of the nodes can be identified by the nodelabels function. Alternatively, you can specify a vector of length two that contains either the number or the names of two tips that coalesce in the node of interest.

If the node subtends a polytomy, any two clades of the the polytomy can be chosen by polytom. On a plotted phylogeny, the clades are numbered from bottom to top and polytom is used to index the two clades one likes to swop.

Value

an object of class "phylo".

Author(s)

Christoph Heibl ⟨heibl@lmu.de⟩

See Also

plot.phylo, nodelabels, root, drop.tip

Examples

```
# create a random tree:
tre <- rtree(25)

# visualize labels of internal nodes:
plot.phylo(tre, use.edge.length=FALSE)
nodelabels()

# rotate clades around node 30:
tre.new <- rotate(tre, 30)

# compare the results:
X11() # open new graphical device
par(mfrow=c(1,2)) # devide graphical device
plot(tre) # plot old tre
plot(tre.new) # plot new tree

# visualize labels of terminal nodes:
X11() # open new graphical device
plot.phylo(tre)
tiplabels()

# rotate clades containing nodes 12 and 20:
tre.new <- rotate(tre, c(12, 21))

# compare the results:
```

```
X11() # open new graphical device
par(mfrow=c(1,2)) # devide graphical device
plot(tre) # plot old tre
plot(tre.new) # plot new tree

# or you migth just specify tiplabel names:
tre.new <- rotate(tre, c("t3", "t14"))

# compare the results:
X11() # open new graphical device
par(mfrow=c(1,2)) # devide graphical device
plot(tre) # plot old tre
plot(tre.new) # plot new tree
```

`rtree` *Generates Random Trees*

Description

These functions generate trees by splitting randomly the edges (`rtree`) or randomly clustering the tips (`rcoal`). `rtree` generates general (non-ultrametric) trees, and `rcoal` generates coalescent (ultrametric) trees.

Usage

```
rtree(n, rooted = TRUE, tip.label = NULL, br = runif, ...)
rcoal(n, tip.label = NULL, br = "coalescent", ...)
rmtree(N, n, rooted = TRUE, tip.label = NULL, br = runif, ...)
```

Arguments

`n`	an integer giving the number of tips in the tree.
`rooted`	a logical indicating whether the tree should be rooted (the default).
`tip.label`	a character vector giving the tip labels; if not specified, the tips "t1", "t2", ..., are given.
`br`	either an R function used to generate the branch lengths (`rtree` or `NULL` to give no branch lengths) or the coalescence times (`rcoal`). For the latter, a genuine coalescent tree is simulated by default.
`...`	further argument(s) to be passed to `br`.
`N`	an integer giving the number of trees to generate.

Details

The trees generated are bifurcating. If `rooted = FALSE` in (`rtree`), the tree is trifurcating at its root.

The default function to generate branch lengths in `rtree` is `runif`. If further arguments are passed to `br`, they need to be tagged (e.g., `min = 0, max = 10`).

`rmtree` calls successively `rtree` and set the class of the returned object appropriately.

Value

An object of class "phylo" or of class "multiPhylo" in the case of rmtree.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

Examples

```
layout(matrix(1:9, 3, 3))
### Nine random trees:
for (i in 1:9) plot(rtree(20))
### Nine random cladograms:
for (i in 1:9) plot(rtree(20, FALSE), type = "c")
### generate 4 random trees of bird orders:
data(bird.orders)
layout(matrix(1:4, 2, 2))
for (i in 1:4)
  plot(rcoal(23, tip.label = bird.orders$tip.label), no.margin = TRUE)
layout(matrix(1))
```

---

seg.sites *Find Segregating Sites in DNA Sequences*

Description

This function gives the indices of segregating (polymorphic) sites in a sample of DNA sequences.

Usage

```
seg.sites(x)
```

Arguments

x	a matrix or a list which contains the DNA sequences.

Details

If the sequences are in a list, all the sequences must be of the same length.

Value

A numeric vector giving the indices of the segregating sites.

Note

The present version looks for the sites which are "variable" in the data in terms of different *letters*. This may give unexpected results if there are ambiguous bases in the data.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`base.freq`, `GC.content`, `theta.s`, `nuc.div`

Examples

```
data(woodmouse)
y <- seg.sites(woodmouse)
y
length(y)
```

`sh.test` *Shimodaira-Hasegawa Test*

Description

This function computes the Shimodaira–Hasegawa test for a set of trees.

Usage

```
sh.test(..., x, model = DNAmodel(), B = 100)
```

Arguments

`...`	either a series of objects of class `"phylo"` separated by commas, or a list containing such objects.
`x`	a list, a matrix, or a data frame containing the (aligned) DNA sequences.
`model`	the model to be fitted to each tree (as an object of `"DNAmodel"`).
`B`	the number of bootstrap replicates.

Details

The present implementation follows the original formulation of Shimodaira and Hasegawa (1999). A difference is that the bootstrap resampling is done on the original sequence data rather than the RELL method as sugested by Shimodaira and Hasegawa.

Value

a numeric vector with the P-value associated with each tree given in `...`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Shimodaira, H. and Hasegawa, M. (1999) Multiple comparisons of log-likelihoods with applications to phylogenetic inference. *Molecular Biology and Evolution*, **16**, 1114–1116.

See Also

`mlphylo`, `DNAmodel`

Examples

```
data(woodmouse)
t1 <- nj(dist.dna(woodmouse))
t2 <- rtree(15, tip.label = t1$tip.label)
t3 <- rtree(15, tip.label = t1$tip.label)
### Are the NJ tree and two random tress significantly different?
## Not run: sh.test(t1, t2, t3, x = woodmouse, B = 100)
```

`skyline` *Skyline Plot Estimate of Effective Population Size*

Description

`skyline` computes the *generalized skyline plot* estimate of effective population size from an estimated phylogeny. The demographic history is approximated by a step-function. The number of parameters of the skyline plot (i.e. its smoothness) is controlled by a parameter `epsilon`.

`find.skyline.epsilon` searches for an optimal value of the `epsilon` parameter, i.e. the value that maximizes the AICc-corrected log-likelihood (`logL.AICc`).

Usage

```
skyline(x, ...)
## S3 method for class 'phylo':
skyline(x, ...)
## S3 method for class 'coalescentIntervals':
skyline(x, epsilon=0, ...)
## S3 method for class 'collapsedIntervals':
skyline(x, old.style=FALSE, ...)
find.skyline.epsilon(ci, GRID=1000, MINEPS=1e-6, ...)
```

Arguments

`x`	Either an ultrametric tree (i.e. an object of class `"phylo"`), or coalescent intervals (i.e. an object of class `"coalescentIntervals"`), or collapsed coalescent intervals (i.e. an object of class `"collapsedIntervals"`).
`epsilon`	collapsing parameter that controls the amount of smoothing (allowed range: from `0` to `ci$total.depth`, default value: 0). This is the same parameter as in collapsed.intervals.

`old.style`	Parameter to choose between two slightly different variants of the generalized skyline plot (Strimmer and Pybus, pers. comm.). The default value `FALSE` is recommended.
`ci`	coalescent intervals (i.e. an object of class `"coalescentIntervals"`)
`GRID`	Parameter for the grid search for `epsilon` in `find.skyline.epsilon`.
`MINEPS`	Parameter for the grid search for `epsilon` in `find.skyline.epsilon`.
`...`	Any of the above parameters.

Details

`skyline` implements the *generalized skyline plot* introduced in Strimmer and Pybus (2001). For `epsilon = 0` the generalized skyline plot degenerates to the *classic skyline plot* described in Pybus et al. (2000). The latter is in turn directly related to lineage-through-time plots (Nee et al., 1995).

Value

`skyline` returns an object of class `"skyline"` with the following entries:

`time`	A vector with the time at the end of each coalescent interval (i.e. the accumulated interval lengths from the beginning of the first interval to the end of an interval)
`interval.length`	A vector with the length of each interval.
`population.size`	A vector with the effective population size of each interval.
`parameter.count`	Number of free parameters in the skyline plot.
`epsilon`	The value of the underlying smoothing parameter.
`logL`	Log-likelihood of skyline plot (see Strimmer and Pybus, 2001).
`logL.AICc`	AICc corrected log-likelihood (see Strimmer and Pybus, 2001).

`find.skyline.epsilon` returns the value of the `epsilon` parameter that maximizes `logL.AICc`.

Author(s)

Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

References

Strimmer, K. and Pybus, O. G. (2001) Exploring the demographic history of DNA sequences using the generalized skyline plot. *Molecular Biology and Evolution*, **18**, 2298–2305.

Pybus, O. G, Rambaut, A. and Harvey, P. H. (2000) An integrated framework for the inference of viral population history from reconstructed genealogies. *Genetics*, **155**, 1429–1437.

Nee, S., Holmes, E. C., Rambaut, A. and Harvey, P. H. (1995) Inferring population history from molecular phylogenies. *Philosophical Transactions of the Royal Society of London. Series B. Biological Sciences*, **349**, 25–31.

See Also

coalescent.intervals, collapsed.intervals, skylineplot, ltt.plot.

Examples

```
# get tree
data("hivtree.newick") # example tree in NH format
tree.hiv <- read.tree(text = hivtree.newick) # load tree

# corresponding coalescent intervals
ci <- coalescent.intervals(tree.hiv) # from tree

# collapsed intervals
cl1 <- collapsed.intervals(ci,0)
cl2 <- collapsed.intervals(ci,0.0119)

#### classic skyline plot ####
sk1 <- skyline(cl1)        # from collapsed intervals
sk1 <- skyline(ci)         # from coalescent intervals
sk1 <- skyline(tree.hiv)   # from tree
sk1

plot(skyline(tree.hiv))
skylineplot(tree.hiv) # shortcut

plot(sk1, show.years=TRUE, subst.rate=0.0023, present.year = 1997)

#### generalized skyline plot ####

sk2 <- skyline(cl2)              # from collapsed intervals
sk2 <- skyline(ci, 0.0119)       # from coalescent intervals
sk2 <- skyline(tree.hiv, 0.0119) # from tree
sk2

plot(sk2)

# classic and generalized skyline plot together in one plot
plot(sk1, show.years=TRUE, subst.rate=0.0023, present.year = 1997, col=c(grey(.8),1))
lines(sk2,  show.years=TRUE, subst.rate=0.0023, present.year = 1997)
legend(.15,500, c("classic", "generalized"), col=c(grey(.8),1),lty=1)

# find optimal epsilon parameter using AICc criterion
find.skyline.epsilon(ci)

sk3 <- skyline(ci, -1) # negative epsilon also triggers estimation of epsilon
sk3$epsilon
```

---

skylineplot *Drawing Skyline Plot Graphs*

Description

These functions provide various ways to draw *skyline plot* graphs on the current graphical device. Note that `skylineplot(z, ...)` is simply a shortcut for `plot(skyline(z, ...))`. The skyline plot itself is an estimate of effective population size through time, and is computed using the function `skyline`.

Usage

```
## S3 method for class 'skyline':
plot(x, show.years=FALSE, subst.rate, present.year, ...)
## S3 method for class 'skyline':
lines(x, show.years=FALSE, subst.rate, present.year, ...)
skylineplot(z, ...)
skylineplot.deluxe(tree, ...)
```

Arguments

`x`	skyline plot data (i.e. an object of class `"skyline"`).
`z`	Either an ultrametric tree (i.e. an object of class `"phylo"`), or coalescent intervals (i.e. an object of class `"coalescentIntervals"`), or collapsed coalescent intervals (i.e. an object of class `"collapsedIntervals"`).
`tree`	ultrametric tree (i.e. an object of class `"phylo"`).
`show.years`	option that determines whether the time is plotted in units of of substitutions (default) or in years (requires specification of substution rate and year of present).
`subst.rate`	substitution rate (see option show.years).
`present.year`	present year (see option show.years).
`...`	further arguments to be passed on to `skyline()` and `plot()`.

Details

See `skyline` for more details (incl. references) about the skyline plot method.

Author(s)

Korbinian Strimmer (`http://www.stat.uni-muenchen.de/~strimmer/`)

See Also

`plot` and `lines` for the basic plotting function in R, `coalescent.intervals`, `skyline`

Examples

```
# get tree
data("hivtree.newick") # example tree in NH format
tree.hiv <- read.tree(text = hivtree.newick) # load tree

#### classic skyline plot
skylineplot(tree.hiv) # shortcut
```

```
#### plot classic and generalized skyline plots and estimate epsilon
sk.opt <- skylineplot.deluxe(tree.hiv)
sk.opt$epsilon

#### classic and generalized skyline plot ####
sk1 <- skyline(tree.hiv)
sk2 <- skyline(tree.hiv, 0.0119)

# use years rather than substitutions as unit for the time axis
plot(sk1, show.years=TRUE, subst.rate=0.0023, present.year = 1997, col=c(grey(.8),1))
lines(sk2,  show.years=TRUE, subst.rate=0.0023, present.year = 1997)
legend(.15,500, c("classic", "generalized"), col=c(grey(.8),1),lty=1)

#### various skyline plots for different epsilons
layout(mat= matrix(1:6,2,3,byrow=TRUE))
ci <- coalescent.intervals(tree.hiv)
plot(skyline(ci, 0.0));title(main="0.0")
plot(skyline(ci, 0.007));title(main="0.007")
plot(skyline(ci, 0.0119),col=4);title(main="0.0119")
plot(skyline(ci, 0.02));title(main="0.02")
plot(skyline(ci, 0.05));title(main="0.05")
plot(skyline(ci, 0.1));title(main="0.1")
layout(mat= matrix(1:1,1,1,byrow=TRUE))
```

`subtreeplot` *Zoom on a Portion of a Phylogeny by Successive Clicks*

Description

This function plots simultaneously a whole phylogenetic tree (supposedly large) and a portion of it determined by clicking on the nodes of the phylogeny. On exit, returns the last subtree visualized.

Usage

```
subtreeplot(x, wait=FALSE, ...)
```

Arguments

`x`	an object of class `"phylo"`.
`wait`	a logical indicating whether the node beeing processed should be printed (useful for big phylogenies).
`...`	further arguments passed to `plot.phylo`.

Details

This function aims at easily exploring very large trees. The main argument is a phylogenetic tree, and the second one is a logical indicating whether a waiting message should be printed while the calculation is being processed.

The whole tree is plotted on the left-hand side in half of the device. The subtree is plotted on the right-hand side in the other half. The user clicks on the nodes in the complete tree and the subtree corresponding to this node is ploted in the right-hand side. There is no limit for the number of clicks that can be done. On exit, the subtree on the right hand side is returned.

To use a subtree as the new tree in which to zoom, the user has to use the function many times. This can however be done in a single command line (see example 2).

Author(s)

Damien de Vienne ⟨damien.de-vienne@u-psud.fr⟩

See Also

`plot.phylo`, `drop.tip`, `subtrees`

Examples

```
## Not run:
#example 1: simple
tree1<-rtree(50) #random tree with 50 leaves
tree2<-subtreeplot(tree1, wait=TRUE) # on exit, tree2 will be a subtree of tree1.

#example 2: more than one zoom
tree1<-rtree(60)
tree2<-subtreeplot(subtreeplot(subtreeplot(tree1))) #allows three succssive zooms.
## End(Not run)
```

`subtrees` *All subtrees of a Phylogenetic Tree*

Description

This function returns a list of all the subtrees of a phylogenetic tree.

Usage

```
subtrees(tree, wait=FALSE)
```

Arguments

`tree`	an object of class `"phylo"`.
`wait`	a logical indicating whether the node beeing processed should be printed (useful for big phylogenies).

Value

`subtrees` returns a list of trees of class `"phylo"` and returns invisibly for each subtree a list with the following components:

`tip.label`

`node.label`

`Ntip`

`Nnode`

Author(s)

Damien de Vienne ⟨damien.de-vienne@u-psud.fr⟩

See Also

`zoom`, `subtreeplot` for functions extracting particular subtrees.

Examples

```
### Random tree with 12 leaves
phy<-rtree(12)
par(mfrow=c(4,3))
plot(phy, sub="Complete tree")

### Extract the subtrees
l<-subtrees(phy)

### plot all the subtrees
for (i in 1:11) plot(l[[i]], sub=paste("Node", l[[i]]$node.label[1]))
par(mfrow=c(1,1))
```

`summary.phylo` *Print Summary of a Phylogeny*

Description

The first function prints a compact summary of a phylogenetic tree (an object of class `"phylo"`). The three other functions return the number of tips, nodes, or edges, respectively.

Usage

```
## S3 method for class 'phylo':
summary(object, ...)
Ntip(phy)
Nnode(phy, internal.only = TRUE)
Nedge(phy)
```

Arguments

`object, phy`	an object of class `"phylo"`.
`...`	further arguments passed to or from other methods.
`internal.only`	a logical indicating whether to return the number of internal nodes only (the default), or of internal and terminal (tips) nodes (if `FALSE`).

Details

The summary includes the numbers of tips and of nodes, summary statistics of the branch lengths (if they are available) with mean, variance, minimum, first quartile, median, third quartile, and maximum, listing of the first ten tip labels, and (if available) of the first ten node labels. It is also printed whether some of these optional elements (branch lengths, node labels, and root edge) are not found in the tree.

If the tree was estimated by maximum likelihood with `mlphylo`, a summary of the model fit and the parameter estimated is printed.

`summary` simply prints its results on the standard output and is not meant for programming.

Value

A NULL value in the case of `summary`, a single numeric value for the three other functions.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`read.tree`, `summary` for the generic R function

Examples

```
data(bird.families)
summary(bird.families)
Ntip(bird.families)
Nnode(bird.families)
Nedge(bird.families)
```

`theta.h` *Population Parameter THETA using Homozygosity*

Description

This function computes the population parameter THETA using the homozygosity (or mean heterozygosity) from gene frequencies.

Usage

```
theta.h(x, standard.error = FALSE)
```

Arguments

`x`	a vector or a factor.
`standard.error`	a logical indicating whether the standard error of the estimated theta should be returned (`TRUE`), the default being `FALSE`.

Details

The argument `x` can be either a factor or a vector. If it is a factor, then it is taken to give the individual alleles in the population. If it is a numeric vector, then its values are taken to be the numbers of each allele in the population. If it is a non-numeric vector, it is a coerced as a factor.

The standard error is computed with an approximation due to Chakraborty and Weiss (1991).

Value

a numeric vector of length one with the estimated theta (the default), or of length two if the standard error is returned (`standard.error = TRUE`).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Zouros, E. (1979) Mutation rates, population sizes and amounts of electrophoretic variation at enzyme loci in natural populations. *Genetics*, **92**, 623–646.

Chakraborty, R. and Weiss, K. M. (1991) Genetic variation of the mitochondrial DNA genome in American Indians is at mutation-drift equilibrium. *American Journal of Human Genetics*, **86**, 497–506.

See Also

`heterozygosity`, `theta.s`, `theta.k`

`theta.k` *Population Parameter THETA using Expected Number of Alleles*

Description

This function computes the population parameter THETA using the expected number of alleles.

Usage

```
theta.k(x, n = NULL, k = NULL)
```

Arguments

`x`	a vector or a factor.
`n`	a numeric giving the sample size.
`k`	a numeric giving the number of alleles.

Details

This function can be used in two ways: either with a vector giving the individual genotypes from which the sample size and number of alleles are derived (`theta.k(x)`), or giving directly these two quantities (`theta.k(n, k)`).

The argument `x` can be either a factor or a vector. If it is a factor, then it is taken to give the individual alleles in the population. If it is a numeric vector, then its values are taken to be the numbers of each allele in the population. If it is a non-numeric vector, it is a coerced as a factor.

Both arguments `n` and `k` must be single numeric values.

Value

a numeric vector of length one with the estimated theta.

Note

For the moment, no standard-error or confidence interval is computed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Ewens, W. J. (1972) The sampling theory of selectively neutral alleles. *Theoretical Population Biology*, **3**, 87–112.

See Also

`theta.h`, `theta.s`

`theta.s` *Population Parameter THETA using Segregating Sites in DNA Sequences*

Description

This function computes the population parameter THETA using the number of segregating sites `s` in a sample of `n` DNA sequences.

Usage

```
theta.s(s, n, variance = FALSE)
```

Arguments

`s`	a numeric giving the number of segregating sites.
`n`	a numeric giving the number of sequences.
`variance`	a logical indicating whether the variance of the estimated THETA should be returned (`TRUE`), the default being `FALSE`.

Value

a numeric vector of length one with the estimated theta (the default), or of length two if the standard error is returned (`variance = TRUE`).

Note

The number of segregating sites needs to be computed beforehand, for instance with the function `seg.sites` (see example below).

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Watterson, G. (1975) On the number of segragating sites in genetical models without recombination. *Theoretical Population Biology*, **7**, 256–276.

Tajima, F. (1989) Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. *Genetics*, **123**, 585–595.

See Also

`theta.h`, `theta.k`, `seg.sites`, `nuc.div`

Examples

```
data(woodmouse)
y <- seg.sites(woodmouse)
s <- length(y)
n <- length(woodmouse)
theta.s(s, n)
theta.s(s, n, variance = TRUE)
```

`unique.multiPhylo` *Revomes Duplicate Trees*

Description

This function scans a list of trees, and returns a list with the duplicate trees removed. By default the labelled topologies are compared.

Usage

```
## S3 method for class 'multiPhylo':
unique(x, incomparables = FALSE,
        use.edge.length = FALSE,
        use.tip.label = TRUE, ...)
```

Arguments

`x`	an object of class `"multiPhylo"`.
`incomparables`	unused (for compatibility with the generic).
`use.edge.length`	a logical specifying whether to consider the edge lengths in the comparisons; the default is `FALSE`.
`use.tip.label`	a logical specifying whether to consider the tip labels in the comparisons; the default is `TRUE`.
`...`	further arguments passed to or from other methods.

Value

an object of class `"multiPhylo"` which is a list of objects of class `"phylo"`.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`all.equal.phylo`, `unique` for the generic R function, `read.tree`, `read.nexus`

Examples

```
TR <- rmtree(50, 4)
length(unique(TR)) # not always 15...
howmanytrees(4)
```

varcomp *Compute Variance Component Estimates*

Description

Get variance component estimates from a fitted `lme` object.

Usage

```
varcomp(x, scale = FALSE, cum = FALSE)
```

Arguments

`x`	A fitted `lme` object
`scale`	Scale all variance so that they sum to 1
`cum`	Send cumulative variance components.

Details

Variance computations is done as in Venables and Ripley's book.

Value

A named vector of class `varcomp` with estimated variance components.

Author(s)

Julien Dutheil ⟨julien.dutheil@univ-montp2.fr⟩

References

Venables, W. N. and Ripley, B. D. (2002) *Modern applied statistics with S (fourth edition).* New York: Springer-Verlag.

See Also

`lme`

Examples

```
data(carnivora)
m <- lme(log10(SW) ~ 1, random = ~ 1|Order/SuperFamily/Family/Genus, data=carnivora)
v <- varcomp(m, TRUE, TRUE)
plot(v)
```

`vcv.phylo` *Phylogenetic Variance-covariance or Correlation Matrix*

Description

This function computes the expected variances and covariances of a continuous phenotype assuming it evolves under a given model (currently only the model of Brownian motion is available).

Usage

```
vcv.phylo(phy, model = "Brownian", cor = FALSE)
```

Arguments

`phy`	an object of class `"phylo"`.
`model`	a character giving the model used to compute the variances and covariances of the phynotype; by default `"Brownian"`. Currently only the Brownian model is available.
`cor`	a logical indicating whether the correlation matrix should be returned (`TRUE`); by default the variance-covariance matrix is returned (`FALSE`).

Value

a numeric matrix with the names of the tips (as given by the `tip.label` of the argument `phy`) as colnames and rownames.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Garland, T. Jr. and Ives, A. R. (2000) Using the past to predict the present: confidence intervals for regression equations in phylogenetic comparative methods. *American Naturalist*, **155**, 346–364.

See Also

`read.tree` to read tree files in Newick format

`weight.taxo` *Define Similarity Matrix*

Description

`weight.taxo` computes a matrix whose entries [i, j] are set to 1 if x[i] == x[j], 0 otherwise.

`weight.taxo2` computes a matrix whose entries [i, j] are set to 1 if x[i] == x[j] AND y[i] != y[j], 0 otherwise.

The diagonal [i, i] is always set to 0.

This returned matrix can be used as a weight matrix in `Moran.I`. `x` and `y` may be vectors of factors.

See further details in `vignette("MoranI")`.

Usage

```
weight.taxo(x)
weight.taxo2(x, y)
```

Arguments

`x, y`	a vector or a factor.

Value

a square numeric matrix.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`Moran.I`, `correlogram.formula`

`which.edge` *Identifies Edges of a Tree*

Description

This function identifies the edges that belong to a group (possibly non-monophyletic) specified as a set of tips.

Usage

```
which.edge(phy, group)
```

Arguments

`phy`	an object of class `"phylo"`.
`group`	a vector of mode numeric or character specifying the tips for which the edges are to be identified.

Details

The group of tips specified in 'group' may be non-monophyletic (paraphyletic or polyphyletic), in which case all edges from the tips to their most recent common ancestor are identified.

The identification is made with the indices of the rows of the matrix 'edge' of the "phylo" object.

Value

a numeric vector.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`bind.tree`, `drop.tip`, `root`

`woodmouse` *Cytochrome b Gene Sequences of Woodmice*

Description

This is a set of 15 sequences of the mitochondrial gene cytochrome *b* of the woodmouse (*Apodemus sylvaticus*) which is a subset of the data analysed by Michaux et al. (2003). The full data set is available through GenBank (accession numbers AJ511877 to AJ511987).

Usage

```
data(woodmouse)
```

Format

The data are stored in an ASCII file using the sequential format for DNA sequences which is read with 'read.dna()'.

Source

Michaux, J. R., Magnanou, E., Paradis, E., Nieberding, C. and Libois, R. (2003) Mitochondrial phylogeography of the Woodmouse (Apodemus sylvaticus) in the Western Palearctic region. *Molecular Ecology*, **12**, 685–697.

See Also

`read.dna`, `DNAbin`, `dist.dna`

Examples

```
data(woodmouse)
str(woodmouse)
```

`write.dna` *Write DNA Sequences in a File*

Description

This function writes in a file a list of DNA sequences in sequential, interleaved, or FASTA format. The names of the vectors of the list are used as taxa names.

Usage

```
write.dna(x, file, format = "interleaved", append = FALSE,
          nbcol = 6, colsep = " ", colw = 10, indent = NULL,
          blocksep = 1)
```

Arguments

`x`	a list or a matrix of DNA sequences.
`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`format`	a character string specifying the format of the DNA sequences. Three choices are possible: `"interleaved"`, `"sequential"`, or `"fasta"`, or any unambiguous abbreviation of these.
`append`	a logical, if `TRUE` the data are appended to the file without erasing the data possibly existing in the file, otherwise the file (if it exists) is overwritten (`FALSE` the default).
`nbcol`	a numeric specifying the number of columns per row (6 by default); may be negative implying that the nucleotides are printed on a single line.
`colsep`	a character used to separate the columns (a single space by default).
`colw`	a numeric specifying the number of nucleotides per column (10 by default).
`indent`	a numeric or a character specifying how the blocks of nucleotides are indented (see details).
`blocksep`	a numeric specifying the number of lines between the blocks of nucleotides (this has an effect only if ‘format = "interleaved"’).

Details

The same three formats are supported in the present function than in read.dna: see its help page and the references below for a description of these formats.

If the sequences have no names, then they are given "1", "2", ... as names in the file.

With the interleaved and sequential formats, the sequences must be all of the same length; if the taxon names are longer than 10 characters, they are truncated and a warning message is issued.

The argument 'indent' specifies how the rows of nucleotides are indented. In the interleaved and sequential formats, the rows with the taxon names are never indented; the subsequent rows are indented with 10 spaces by default (i.e. if 'indent = NULL)'. In the FASTA format, the rows are not indented by default. This default behaviour can be modified by specifying a value to 'indent': the rows are then indented with 'indent' (if it is a character) or 'indent' spaces (if it is a numeric). For example, specifying 'indent = " "' or 'indent = 3' will have exactly the same effect (use 'indent = ""̂' for a tabulation).

Value

None (invisible 'NULL').

Note

Specifying a negative value for 'nbcol' (meaning that the nucleotides are printed on a single line) gives the same result for the interleaved and sequential formats.

The different options are intended to give flexibility in formatting the sequences. For instance, if the sequences are very long it may be judicious to remove all the spaces beween columns '(colsep = ""), in the margins (indent = 0), and between the blocks (blocksep = 0) to produce a smaller file.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Anonymous. FASTA format description. http://www.ncbi.nlm.nih.gov/BLAST/fasta.html

Anonymous. IUPAC ambiguity codes. http://www.ncbi.nlm.nih.gov/SNP/iupac.html

Felsenstein, J. (1993) Phylip (Phylogeny Inference Package) version 3.5c. Department of Genetics, University of Washington. http://evolution.genetics.washington.edu/phylip/phylip.html

See Also

read.dna, read.GenBank

`write.nexus` *Write Tree File in Nexus Format*

Description

This function writes trees in a file with the NEXUS format.

Usage

```
write.nexus(..., file = "", translate = TRUE, original.data = TRUE)
```

Arguments

`...`	either (i) a single object of class `"phylo"`, (ii) a series of such objects separated by commas, or (iii) a list containing such objects.
`file`	a file name specified by either a variable of mode character, or a double-quoted string; if `file = ""` (the default) then the tree is written on the standard output connection.
`translate`	a logical, if `TRUE` (the default) a translation of the tip labels is done which are replaced in the parenthetic representation with tokens.
`original.data`	a logical, if `TRUE` (the default) the data in the original NEXUS file are eventually written in `"file"` (see below).

Details

If `original.data = TRUE`, the file as specified by the attribute `"origin"` of the first tree is read and its data (except the trees) are written in `file`.

If several trees are given, they must have all the same tip labels.

If among the objects given some are not trees of class `"phylo"`, they are simply skipped and not written to the file.

Value

None (invisible 'NULL').

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) NEXUS: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

See Also

`read.nexus`, `read.tree`, `write.tree`, `read.nexus.data`, `write.nexus.data`

`write.nexus.data` *Write Character Data In NEXUS Format*

Description

This function writes in a file a list of sequences in the NEXUS format.The names of the vectors of the list are used as taxon names.

Usage

```
write.nexus.data(x, file, format = "dna", datablock = TRUE,
                 interleaved = TRUE, charsperline = NULL,
                 gap = NULL, missing = NULL)
```

Arguments

`x`	a list of sequences each made of a single vector of mode character where each element is a character state (e.g. "A", "C", ...).
`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`format`	a character string specifying the format of the sequences. Two choices are possible: `dna`, or `protein`, or any unambiguous abbreviation of these. Default is "`dna`".
`datablock`	a logical, if `TRUE` the data are written in a single DATA block. If `FALSE` data is written in TAXA and CHARACTER blocks. Default is `TRUE`.
`interleaved`	a logical, if `TRUE` the data is written in interleaved format with number of characters per line as specified with `charsperline = numerical_value`. If `FALSE`, data is written in sequential format. Default is `TRUE`.
`charsperline`	a numeric specifying the number of characters per line when used with `interleaved = TRUE`. Default is `80`.
`gap`	a character specifying the symbol for gap. Default is "`-`".
`missing`	a character specifying the symbol for missing data. Default is "`?`".

Details

If the sequences have no names, then they are given "1", "2", ..., as names in the file.

Sequences must be all of the same length (i.e., aligned).

Default symbols for missing data and gaps can be changed by using the `missing` and `gap` commands.

Please see files 'data.nex' and 'taxacharacters.nex' for examples of output formats.

Value

None (invisible 'NULL').

Note

...

Author(s)

Johan Nylander ⟨nylander@scs.fsu.edu⟩

References

Maddison, D. R., Swofford, D. L. and Maddison, W. P. (1997) NEXUS: an extensible file format for systematic information. *Systematic Biology*, **46**, 590–621.

See Also

read.nexus,write.nexus, read.nexus.data

Examples

```
## Not run:
## Write interleaved DNA data with 100 characters per line in a DATA block
## Not run: data("woodmouse")
## Not run: write.nexus.data(woodmouse, file= "woodmouse.example.nex", interleaved = TRUE, c
## Write sequential DNA data in TAXA and CHARACTERS blocks
## Not run: data("cynipids")
## Not run: write.nexus.data(cynipids, file= "cynipids.example.nex", format = "protein", dat
## Not run:
```

---

write.tree *Write Tree File in Parenthetic Format*

Description

This function writes in a file a tree in parenthetic format using the Newick (also known as New Hampshire) format.

Usage

```
write.tree(phy, file = "", append = FALSE,
           digits = 10)
```

Arguments

`phy`	an object of class `"phylo"`.
`file`	a file name specified by either a variable of mode character, or a double-quoted string; if `file = ""` (the default) then the tree is written on the standard output connection (i.e. the console).

`append`	a logical, if `TRUE` the tree is appended to the file without erasing the data possibly existing in the file, otherwise the file (if it exists) is overwritten (`FALSE` the default).
`digits`	a numeric giving the number of digits used for printing branch lengths.

Details

The node labels and the root edge length, if available, are written in the file.

Value

a vector of mode character if `file = ""`, none (invisible 'NULL') otherwise.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Felsenstein, J. The Newick tree format. `http://evolution.genetics.washington.edu/phylip/newicktree.html`

Olsen, G. Interpretation of the "Newick's 8:45" tree format standard. `http://evolution.genetics.washington.edu/phylip/newick_doc.html`

See Also

`read.tree`, `read.nexus`, `write.nexus`

`xenarthra` *Molecular Phylogeny of Living Xenarthrans*

Description

This phylogeny was inferred by maximum likelihood analysis of the nuclear gene BRCA1 (breast cancer susceptibility, 2788 sites) sequences for 47 placental and 3 marsupial taxa.

Usage

```
data(xenarthra)
```

Format

The data are stored as an object of class `"phylo"` which structure is described in the help page of the function `read.tree`.

Source

Delsuc, F., Scally, M., Madsen, O., Stanhope, M. J., de Jong, W. W., Catzeflis, F. M., Springer, M. S. and Douzery, E. J. P. (2002) Molecular phylogeny of living xenarthrans and the impact of character and taxon sampling on the placental tree rooting. *Molecular Biology and Evolution*, **19**, 1656–1671.

See Also

`read.tree`

Examples

```
data(xenarthra)
plot(xenarthra)
### remove the margins...
plot(xenarthra, no.margin = TRUE)
### ... and use a smaller font size
plot(xenarthra, no.margin = TRUE, cex = 0.8)
```

---

yule *Fits Yule Model to a Phylogenetic Tree*

Description

This function fits by maximum likelihood a Yule model, i.e. a birth-only model to the branching times computed from a phylogenetic tree.

Usage

```
yule(phy, use.root.edge = FALSE)
```

Arguments

`phy`	an object of class `"phylo"`.
`use.root.edge`	a logical specifying whether to consider the root edge in the calculations.

Details

The tree must be fully dichotomous.

The maximum likelihood estimate of the speciation rate is obtained by the ratio of the number of speciation events on the cumulative number of species through time; these two quantities are obtained with the number of nodes in the tree, and the sum of the branch lengths, respectively.

If there is a ‘root.edge’ element in the phylogenetic tree, and `use.root.edge = TRUE`, then it is assumed that it has a biological meaning and is counted as a branch length, and the root is counted as a speciation event; otherwise the number of speciation events is the number of nodes - 1.

The standard-error of lambda is computed with the second derivative of the log-likelihood function.

Value

An object of class "yule" which is a list with the following components:

`lambda`	the maximum likelihood estimate of the speciation (birth) rate.
`se`	the standard-error of lambda.
`loglik`	the log-likelihood at its maximum.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

`branching.times`, `diversi.gof`, `diversi.time`, `ltt.plot`, `birthdeath`, `bd.ext`, `yule.cov`

`yule.cov` *Fits the Yule Model With Covariates*

Description

This function fits by maximum likelihood the Yule model with covariates, that is a birth-only model where speciation rate is determined by a generalized linear model.

Usage

```
yule.cov(phy, formula, data = NULL)
```

Arguments

`phy`	an object of class `"phylo"`.
`formula`	a formula specifying the model to be fitted.
`data`	the name of the data frame where the variables in `formula` are to be found; by default, the variables are looked for in the global environment.

Details

The model fitted is a generalization of the Yule model where the speciation rate is determined by:

$$\ln \frac{\lambda_i}{1-\lambda_i} = \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \alpha$$

where λ_i is the speciation rate for species i, $x_{i1}, x_{i2}, \ldots$ are species-specific variables, and $\beta_1, \beta_2, \ldots, \alpha$ are parameters to be estimated. The term on the left-hand side above is a logit function often used in generalized linear models for binomial data (see `family`). The above model can be written in matrix form:

$$\text{logit}\lambda_i = x_i'\beta$$

The standard-errors of the parameters are computed with the second derivatives of the log-likelihood function. (See References for other details on the estimation procedure.)

The function needs three things:

a phylogenetic tree which may contain multichotomies;

a formula which specifies the predictors of the model described above: this is given as a standard R formula and has no response (no left-hand side term), for instance: `~ x + y`, it can include interactions (`~ x + a * b`) (see `formula` for details);

the predictors specified in the formula must be accessible to the function (either in the global space, or though the `data` option); they can be numeric vectors or factors. The length and the order of these data are important: the number of values (length) must be equal to the number of tips of the tree + the number of nodes. The order is the following: first the values for the tips in the same order than for the labels, then the values for the nodes sequentially from the root to the most terminal nodes (i.e. in the order given by `phy$edge`).

The user must obtain the values for the nodes separately.

Note that the method in its present implementation assumes that the change in a species trait is more or less continuous between two nodes or between a node and a tip. Thus reconstructing the ancestral values with a Brownian motion model may be consistent with the present method. This can be done with the function `pic` but currently needs some hacking!

Value

A NULL value is returned, the results are simply printed.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

References

Paradis, E. (2005) Statistical analysis of diversification with species traits. *Evolution*, **59**, 1–12.

See Also

`branching.times`, `diversi.gof`, `diversi.time`, `ltt.plot`, `birthdeath`, `bd.ext`, `yule`

Examples

```
### a simple example with some random data
data(bird.orders)
x <- rnorm(45) # the tree has 23 tips and 22 nodes
### the standard-error for x should be as large as
### the estimated parameter
```

```
yule.cov(bird.orders, ~ x)
### compare with the simple Yule model, eventually
### with a likelihood ratio test
yule(bird.orders)
### another example with a tree that has a multichotomy
### but we cannot run yule() because of this!
data(bird.families)
y <- rnorm(272) # 137 tips + 135 nodes
yule.cov(bird.families, ~ y)
```

`zoom` *Zoom on a Portion of a Phylogeny*

Description

This function plots simultaneously a whole phylogenetic tree (supposedly large) and a portion of it.

Usage

```
zoom(phy, focus, subtree = FALSE, col = rainbow, ...)
```

Arguments

`phy`	an object of class `"phylo"`.
`focus`	a vector, either numeric or character, or a list of vectors specifying the tips to be focused on.
`subtree`	a logical indicating whether to show the context of the extracted subtrees.
`col`	a vector of colours used to show where the subtrees are in the main tree, or a function .
`...`	further arguments passed to `plot.phylo`.

Details

This function aims at exploring very large trees. The main argument is a phylogenetic tree, and the second one is a vector or a list of vectors specifying the tips to be focused on. The vector(s) can be either numeric and thus taken as the indices of the tip labels, or character in which case it is taken as the corresponding tip labels.

The whole tree is plotted on the left-hand side in a narrower sub-window (about a quarter of the device) without tip labels. The subtrees consisting of the tips in 'focus' are extracted and plotted on the right-hand side starting from the top left corner and successively column-wise.

If the argument 'col' is a vector of colours, as many colours as the number of subtrees must be given. The alternative is to give a function that will create colours or grey levels from the number of subtrees: see `rainbow` for some possibilities with colours.

Author(s)

Emmanuel Paradis ⟨Emmanuel.Paradis@mpl.ird.fr⟩

See Also

plot.phylo, drop.tip, layout, rainbow, grey

Examples

```
## Not run:
data(chiroptera)
zoom(chiroptera, 1:20, subtree = TRUE)
zoom(chiroptera, grep("Plecotus", chiroptera$tip.label))
zoom(chiroptera, list(grep("Plecotus", chiroptera$tip.label),
                      grep("Pteropus", chiroptera$tip.label)))
## End(Not run)
```

Index