Package: ape (via r-universe)

October 28, 2026

Version 5.8-1

Date 2024-12-10

Title Analyses of Phylogenetics and Evolution

Depends R (>= 3.2.0)

Suggests gee, expm, igraph, phangorn, xml2

Imports nlme, lattice, graphics, methods, stats, utils, parallel, Rcpp (>= 0.12.0), digest

LinkingTo Rcpp

ZipData no

Description Functions for reading, writing, plotting, and manipulating phylogenetic trees, analyses of comparative data in a phylogenetic framework, ancestral character analyses, analyses of diversification and macroevolution, computing distances from DNA sequences, reading and writing nucleotide sequences as well as importing from BioConductor, and several tools such as Mantel's test, generalized skyline plots, graphical exploration of phylogenetic data (alex, trex, kronoviz), estimation of absolute evolutionary rates and clock-like trees using mean path lengths and penalized likelihood, dating trees with non-contemporaneous sequences, translating DNA into AA sequences, and assessing sequence alignments. Phylogeny estimation can be done with the NJ, BIONJ, ME, MVR, SDM, and triangle methods, and several methods handling incomplete distance matrices (NJ*, BIONJ*, MVR*, and the corresponding triangle method). Some functions call external applications (PhyML, Clustal, T-Coffee, Muscle) whose results are returned into R.

License GPL-2 | GPL-3

URL https://github.com/emmanuelparadis/ape

BugReports https://github.com/emmanuelparadis/ape/issues

Encoding UTF-8

NeedsCompilation yes

Author Emmanuel Paradis [aut, cre, cph] (<https://orcid.org/0000-0003-3092-2199>), Simon Blomberg [aut, cph] (<https://orcid.org/0000-0003-1062-0839>), Ben Bolker [aut, cph] (<https://orcid.org/0000-0002-2127-0443>), Joseph Brown [aut, cph] (<https://orcid.org/0000-0002-3835-8062>), Santiago Claramunt [aut, cph] (<https://orcid.org/0000-0002-8926-5974>), Julien Claude [aut, cph] (<https://orcid.org/0000-0002-9267-1228>), Hoa Sien Cuong [aut, cph], Richard Desper [aut, cph], Gilles Didier [aut, cph] (<https://orcid.org/0000-0003-0596-9112>), Benoit Durand [aut, cph], Julien Dutheil [aut, cph] (<https://orcid.org/0000-0001-7753-4121>), RJ Ewing [aut, cph], Olivier Gascuel [aut, cph], Thomas Guillerme [aut, cph] (<https://orcid.org/0000-0003-4325-1275>), Christoph Heibl [aut, cph] (<https://orcid.org/0000-0002-7655-3299>), Anthony Ives [aut, cph] (<https://orcid.org/0000-0001-9375-9523>), Bradley Jones [aut, cph] (<https://orcid.org/0000-0003-4498-1069>), Franz Krah [aut, cph] (<https://orcid.org/0000-0001-7866-7508>), Daniel Lawson [aut, cph] (<https://orcid.org/0000-0002-5311-6213>), Vincent Lefort [aut, cph], Pierre Legendre [aut, cph] (<https://orcid.org/0000-0002-3838-3305>), Jim Lemon [aut, cph], Guillaume Louvel [aut, cph] (<https://orcid.org/0000-0002-7745-0785>), Federico Marotta [aut, cph], Eric Marcon [aut, cph] (<https://orcid.org/0000-0002-5249-321X>), Rosemary McCloskey [aut, cph] (<https://orcid.org/0000-0002-9772-8553>), Johan Nylander [aut, cph], Rainer Opgen-Rhein [aut, cph], Andrei-Alin Popescu [aut, cph], Manuela Royer-Carenzi [aut, cph], Klaus Schliep [aut, cph] (<https://orcid.org/0000-0003-2941-0161>), Korbinian Strimmer [aut, cph] (<https://orcid.org/0000-0001-7917-2056>), Damien de Vienne [aut, cph] (<https://orcid.org/0000-0001-9532-5251>)

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

Repository https://cran.r-universe.dev

Date/Publication 2024-12-16 02:14:52 UTC

RemoteUrl https://github.com/cran/ape

RemoteRef HEAD

RemoteSha 82676497f56768c119ce5ba600e7c8db9f4ca486

Contents

ape-package 7
AAbin 7
ace 10
add.scale.bar 15

additive . . . 16
alex . . . 17
all.equal.DNAbin . . . 18
all.equal.phylo . . . 19
alview . . . 21
apetools . . . 22
as.alignment . . . 23
as.bitsplits . . . 25
as.matching . . . 26
as.phylo . . . 28
as.phylo.formula . . . 30
axisPhylo . . . 31
balance . . . 32
base.freq . . . 33
bd.ext . . . 34
bd.time . . . 35
binaryPGLMM . . . 37
bind.tree . . . 43
BIONJ . . . 46
bird.families . . . 47
bird.orders . . . 48
birthdeath . . . 48
boot.phylo . . . 50
branching.times . . . 53
c.phylo . . . 54
CADM.global . . . 55
carnivora . . . 59
checkAlignment . . . 60
checkLabel . . . 61
checkValidPhylo . . . 62
cherry . . . 62
chiroptera . . . 63
chronoMPL . . . 64
chronopl . . . 66
chronos . . . 68
clustal . . . 70
coalescent.intervals . . . 73
collapse.singles . . . 74
collapsed.intervals . . . 75
compar.cheverud . . . 77
compar.gee . . . 78
compar.lynch . . . 81
compar.ou . . . 82
comparePhylo . . . 84
compute.brlen . . . 86
compute.brtime . . . 87
consensus . . . 88
cophenetic.phylo . . . 89

cophyloplot . . . 90
corBlomberg . . . 92
corBrownian . . . 93
corClasses . . . 94
corGrafen . . . 95
corMartins . . . 97
corPagel . . . 98
corphylo . . . 100
correlogram.formula . . . 105
data.nex . . . 107
dbd . . . 107
def . . . 109
degree . . . 111
del.gaps . . . 112
delta.plot . . . 113
dist.dna . . . 114
dist.gene . . . 117
dist.topo . . . 118
diversi.gof . . . 120
diversi.time . . . 121
diversity.contrast.test . . . 123
DNAbin . . . 124
DNAbin2indel . . . 127
dnds . . . 127
drop.tip . . . 129
edges . . . 131
evonet . . . 132
ewLasso . . . 134
FastME . . . 135
gammaStat . . . 137
getAnnotationsGenBank . . . 138
hivtree . . . 139
howmanytrees . . . 140
identify.phylo . . . 141
image.DNAbin . . . 143
Initialize.corPhyl . . . 144
is.binary . . . 145
is.compatible . . . 146
is.monophyletic . . . 147
is.ultrametric . . . 148
kronoviz . . . 149
label2table . . . 150
ladderize . . . 151
latag2n . . . 152
lmorigin . . . 153
LTT . . . 155
ltt.plot . . . 157
makeLabel . . . 160

makeNodeLabel . . . 161
mantel.test . . . 163
mat3 . . . 164
mat5M3ID . . . 165
mat5Mrand . . . 165
matexpo . . . 166
mcconwaysims.test . . . 167
mcmc.popsize . . . 168
mixedFontLabel . . . 170
Moran.I . . . 172
MPR . . . 173
mrca . . . 175
mst . . . 176
multi2di . . . 177
multiphylo . . . 179
mvr . . . 180
nj . . . 181
njs . . . 182
node.dating . . . 183
node.depth . . . 185
nodelabels . . . 186
nodepath . . . 190
parafit . . . 191
pcoa . . . 193
phydataplot . . . 196
phymltest . . . 200
pic . . . 203
pic.ortho . . . 204
plot.correlogram . . . 205
plot.phylo . . . 206
plot.phylo.extra . . . 212
plot.varcomp . . . 213
plotTreeTime . . . 214
print.phylo . . . 215
rDNAbin . . . 216
read.caic . . . 217
read.dna . . . 218
read.GenBank . . . 223
read.gff . . . 224
read.nexus . . . 226
read.nexus.data . . . 227
read.tree . . . 229
reconstruct . . . 231
reorder.phylo . . . 233
richness.yule.test . . . 235
rlineage . . . 236
root . . . 238
rotate . . . 240

rTraitCont . . . 242
rTraitDisc . . . 244
rTraitMult . . . 246
rtree . . . 248
rtt . . . 249
SDM . . . 251
seg.sites . . . 252
skyline . . . 253
skylineplot . . . 255
slowinskiguyer.test . . . 257
solveAmbiguousBases . . . 258
speciesTree . . . 259
stree . . . 260
subtreeplot . . . 261
subtrees . . . 262
summary.phylo . . . 263
trans . . . 265
treePop . . . 266
trex . . . 267
triangMtd . . . 268
unique.multiPhylo . . . 269
updateLabel . . . 270
varcomp . . . 272
varCompPhylip . . . 273
vcv . . . 274
vcv2phylo . . . 276
weight.taxo . . . 277
where . . . 277
which.edge . . . 278
woodmouse . . . 279
write.dna . . . 280
write.nexus . . . 281
write.nexus.data . . . 283
write.phyloXML . . . 284
write.tree . . . 285
yule . . . 287
yule.cov . . . 288
yule.time . . . 289
zoom . . . 291

Index **293**

`ape-package` *Analyses of Phylogenetics and Evolution*

Description

ape provides functions for reading, writing, manipulating, analysing, and simulating phylogenetic trees and DNA sequences, computing DNA distances, translating into AA sequences, estimating trees with distance-based methods, and a range of methods for comparative analyses and analysis of diversification. Functionalities are also provided for programming new phylogenetic methods.

The complete list of functions can be displayed with `library(help = ape)`.

More information on **ape** can be found at https://emmanuelparadis.github.io.

Author(s)

Emmanuel Paradis, Ben Bolker, Julien Claude, Hoa Sien Cuong, Richard Desper, Benoit Durand, Julien Dutheil, Olivier Gascuel, Christoph Heibl, Daniel Lawson, Vincent Lefort, Pierre Legendre, Jim Lemon, Yvonnick Noel, Johan Nylander, Rainer Opgen-Rhein, Andrei-Alin Popescu, Klaus Schliep, Korbinian Strimmer, Damien de Vienne

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

References

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

Paradis, E., Claude, J. and Strimmer, K. (2004) APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, **20**, 289–290.

Popescu, A.-A., Huber, K. T. and Paradis, E. (2012) ape 3.0: new tools for distance based phylogenetics and evolutionary analysis in R. *Bioinformatics*, **28**, 1536–1537.

Paradis, E. and Schliep, K. (2019) ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, **35**, 526–528.

`AAbin` *Amino Acid Sequences*

Description

These functions help to create and manipulate AA sequences.

Usage

```
## S3 method for class 'AAbin'
print(x, ...)

## S3 method for class 'AAbin'
x[i, j, drop = FALSE]

## S3 method for class 'AAbin'
c(..., recursive = FALSE)

## S3 method for class 'AAbin'
rbind(...)
## S3 method for class 'AAbin'
cbind(..., check.names = TRUE, fill.with.Xs = FALSE,
             quiet = FALSE)

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

## S3 method for class 'AAbin'
labels(object, ...)

## S3 method for class 'AAbin'
image(x, what, col, bg = "white", xlab = "", ylab = "",
      show.labels = TRUE, cex.lab = 1, legend = TRUE, grid = FALSE,
      show.aa = FALSE, aa.cex = 1, aa.font = 1, aa.col = "black",
      scheme = "Ape_AA",...)

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

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

## S3 method for class 'AAString'
as.AAbin(x, ...)

## S3 method for class 'AAStringSet'
as.AAbin(x, ...)

## S3 method for class 'AAMultipleAlignment'
as.AAbin(x, ...)

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

## S3 method for class 'AAbin'
```

```
as.matrix(x, ...)

## S3 method for class 'AAbin'
as.phyDat(x, ...)

dist.aa(x, pairwise.deletion = FALSE, scaled = FALSE)
AAsubst(x)
```

Arguments

x, object	an object of class "AAbin" (or else depending on the function).
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 returned object is of the lowest possible dimension.
recursive	logical; whether to go down lists and concatenate its elements.
check.names	a logical specifying whether to check the rownames before binding the columns (see details).
fill.with.Xs	a logical indicating whether to keep all possible individuals as indicating by the rownames, and eventually filling the missing data with insertion gaps (ignored if check.names = FALSE).
quiet	a logical to switch off warning messages when some rows are dropped.
what	a vector of characters specifying the amino acids to visualize. Currently, the only possible choice is to show the three categories hydrophobic, small, and hydrophilic.
col	a vector of colours. If missing, this is set to "red", "yellow" and "blue".
bg	the colour used for AA codes not among what (typically X and *).
xlab	the label for the *x*-axis; none by default.
ylab	Idem for the *y*-axis. Note that by default, the labels of the sequences are printed on the *y*-axis (see next option).
show.labels	a logical controlling whether the sequence labels are printed (TRUE by default).
cex.lab	a single numeric controlling the size of the sequence labels. Use cex.axis to control the size of the annotations on the *x*-axis.
legend	a logical controlling whether the legend is plotted (TRUE by default).
grid	a logical controlling whether to draw a grid (FALSE by default).
show.aa	a logical controlling whether to show the AA symbols (FALSE by default).
aa.cex, aa.font, aa.col	control the aspect of the AA symbols (ignored if the previous is FALSE).
scheme	a predefined color scheme. For amino acid options are "Ape_AA", "Zappo_AA", "Clustal" and "Hydrophobicity", for nucleotides "Ape_NT" and "RY_NT".
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.

scaled	a logical value specifying whether to scale the number of AA differences by the sequence length.
...	further arguments to be passed to or from other methods.

Details

These functions help to manipulate amino acid sequences of class "AAbin". These objects are stored in vectors, matrices, or lists which can be manipulated with the usual [operator.

There is a conversion function to and from characters.

The function dist.aa computes the number of AA differences between each pair of sequences in a matrix; this can be scaled by the sequence length. See the function dist.ml in **phangorn** for evolutionary distances with AA sequences.

The function AAsubst returns the indices of the polymorphic sites (similar to seg.sites for DNA sequences; see examples below).

The two functions cbind.AAbin and rbind.AAbin work in the same way than the similar methods for the class "DNAbin": see cbind.DNAbin for more explanations about their respective behaviours.

Value

an object of class "AAbin", "character", "dist", or "numeric", depending on the function.

Author(s)

Emmanuel Paradis, Franz Krah

See Also

read.FASTA, trans, alview

Examples

```
data(woodmouse)
AA <- trans(woodmouse, 2)
seg.sites(woodmouse)
AAsubst(AA)
```

---

ace *Ancestral Character Estimation*

Description

ace estimates ancestral character states, and the associated uncertainty, for continuous and discrete characters. If marginal = TRUE, a marginal estimation procedure is used. With this method, the likelihood values at a given node are computed using only the information from the tips (and branches) descending from this node.

The present implementation of marginal reconstruction for discrete characters does not calculate the most likely state for each node, integrating over all the possible states, over all the other nodes in the tree, in proportion to their probability. For more details, see the Note below.

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

`anova` is another generic function which 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 = if (type == "continuous")
   "REML" else "ML", CI = TRUE,
    model = if (type == "continuous") "BM" else "ER",
    scaled = TRUE, kappa = 1, corStruct = NULL, ip = 0.1,
    use.expm = FALSE, use.eigen = TRUE, marginal = FALSE)
## S3 method for class 'ace'
print(x, digits = 4, ...)
## 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; an object of class `"ace"` in the case of `print`.
`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. Four choices are possible: `"ML"`, `"REML"`, `"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).
use.expm	a logical specifying whether to use the package **expm** to compute the matrix exponential (relevant only if type = "d"). If FALSE, the function matexpo from **ape** is used (see details). This option is ignored if use.eigen = TRUE (see next).
use.eigen	a logical (relevant if type = "d"); if TRUE then the probability matrix is computed with an eigen decomposition instead of a matrix exponential (see details).
marginal	a logical (relevant if type = "d"). By default, the joint reconstruction of the ancestral states are done. Set this option to TRUE if you want the marginal reconstruction (see details.)
digits	the number of digits to be printed.
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 residual maximum likelihood (the default), maximum likelihood (Felsenstein 1973, 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 last case, the specification of phy and model are actually ignored: it is instead given through a correlation structure with the option corStruct.

In the setting method = "ML" and model = "BM" (this used to be the default until **ape** 3.0-7) the maximum likelihood estimation is done simultaneously on the ancestral values and the variance of the Brownian motion process; these estimates are then used to compute the confidence intervals in the standard way. The REML method first estimates the ancestral value at the root (aka, the phylogenetic mean), then the variance of the Brownian motion process is estimated by optimizing the residual log-likelihood. The ancestral values are finally inferred from the likelihood function giving these two parameters. If method = "pic" or "GLS", the confidence intervals are computed using the expected variances under the model, so they depend only on the tree.

It could be shown that, with a continous character, REML results in unbiased estimates of the variance of the Brownian motion process while ML gives a downward bias. Therefore the former is recommanded.

For discrete characters (type = "discrete"), only maximum likelihood estimation is available (Pagel 1994) (see MPR for an alternative method). 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.

By default, the likelihood of the different ancestral states of discrete characters are computed with a joint estimation procedure using a procedure similar to the one described in Pupko et al. (2000). If `marginal = TRUE`, a marginal estimation procedure is used (this was the only choice until **ape** 3.1-1). With this method, the likelihood values at a given node are computed using only the information from the tips (and branches) descending from this node. With the joint estimation, all information is used for each node. The difference between these two methods is further explained in Felsenstein (2004, pp. 259-260) and in Yang (2006, pp. 121-126). The present implementation of the joint estimation uses a "two-pass" algorithm which is much faster than stochastic mapping while the estimates of both methods are very close.

With discrete characters it is necessary to compute the exponential of the rate matrix. The only possibility until **ape** 3.0-7 was the function `matexpo` in **ape**. If `use.expm = TRUE` and `use.eigen = FALSE`, the function `expm`, in the package of the same name, is used. `matexpo` is faster but quite inaccurate for large and/or asymmetric matrices. In case of doubt, use the latter. Since **ape** 3.0-10, it is possible to use an eigen decomposition avoiding the need to compute the matrix exponential; see details in Lebl (2013, sect. 3.8.3). This is much faster and is now the default.

Since version 5.2 of **ape**, `ace` can take state uncertainty for discrete characters into account: this should be coded with R's `NA` only. More details:

https://www.mail-archive.com/r-sig-phylo@r-project.org/msg05286.html

Value

an object of class `"ace"` 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.

Note

Liam Revell points out that for discrete characters the ancestral likelihood values returned with `marginal = FALSE` are actually the marginal estimates, while setting `marginal = TRUE` returns the conditional (scaled) likelihoods of the subtree:

http://blog.phytools.org/2015/05/about-how-acemarginaltrue-does-not.html

Author(s)

Emmanuel Paradis, Ben Bolker

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. (1973) Maximum likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics*, **25**, 471–492.

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

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

Lebl, J. (2013) *Notes on Diffy Qs: Differential Equations for Engineers*. https://www.jirka.org/diffyqs/.

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.

Pupko, T., Pe'er, I, Shamir, R., and Graur, D. (2000) A fast algorithm for joint reconstruction of ancestral amino acid sequences. *Molecular Biology and Evolution*, **17**, 890–896.

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

Yang, Z. (2006) *Computational Molecular Evolution*. Oxford: Oxford University Press.

See Also

`MPR`, `corBrownian`, `compar.ou`, `anova`

Reconstruction of ancestral sequences can be done with the package **phangorn** (see function `?ancestral.pml`).

Examples

```
### 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)
```

`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, y, length = NULL, ask = FALSE,
              lwd = 1, lcol = "black", ...)
```

Arguments

`x`	x location of the bar (can be left missing).
`y`	y location of the bar (can be left missing).
`length`	a numeric value giving the length of the scale bar. If none is supplied, a value is calculated from the data.
`ask`	a logical; if `TRUE` the user is asked to click where to draw the bar. The default is `FALSE`.
`lwd`	the width of the bar.
`lcol`	the colour of the bar (use `col` for the colour of the text).
`...`	further arguments to be passed to `text`.

Details

By default, the bar is placed in a corner of the graph depending on the direction of the tree. Otherwise both `x` and `y` must be specified (if only one is given it is ignored).

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

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(1)
```

`additive` *Incomplete Distance Matrix Filling*

Description

Fills missing entries from incomplete distance matrix using the additive or the ultrametric procedure (see reference for details).

Usage

```
additive(X)
ultrametric(X)
```

Arguments

`X`	a distance matrix or an object of class `"dist"`.

Value

a distance matrix.

Author(s)

Andrei Popescu

References

Makarenkov, V. and Lapointe, F.-J. (2004) A weighted least-squares approach for inferring phylogenies from incomplete distance matrices. *Bioinformatics*, **20**, 2113–2121.

alex *Alignment Explorer With Multiple Devices*

Description

This function helps to explore DNA alignments by zooming in. The user clicks twice defining the opposite corners of the portion which is extracted and drawned on a new window.

Usage

```
alex(x, ...)
```

Arguments

x	an object of class "DNAbin".
...	further arguments to pass to image.DNAbin.

Details

This function works with a DNA alignment (freshly) plotted on an interactive graphical device (i.e., not a file) with image. After calling alex, the user clicks twice defining a rectangle in the alignment, then this portion of the alignment is extacted and plotted on a *new* window. The user can click as many times on the alignment. The process is stopped by a right-click. If the user clicks twice outside the alignment, a message "Try again!" is printed.

Each time alex is called, the alignment is plotted on a new window without closing or deleting those possibly already plotted.

In all cases, the device where x is plotted is the active window after the operation. It should *not* be closed during the whole process.

Value

NULL

Author(s)

Emmanuel Paradis

See Also

image.DNAbin, trex, alview

Examples

```
## Not run:
data(woodmouse)
image(woodmouse)
alex(woodmouse)

## End(Not run)
```

`all.equal.DNAbin` *Compare DNA Sets*

Description

Comparison of DNA sequence sets, particularly when aligned.

Usage

```
## S3 method for class 'DNAbin'
all.equal(target, current, plot = FALSE, ...)
```

Arguments

`target`, `current`	the two sets of sequences to be compared.
`plot`	a logical value specifying whether to plot the sites that are different (only if the labels of both alignments are the same).
`...`	further arguments passed to `image.DNAbin`.

Details

If the two sets of DNA sequences are exactly identical, this function returns `TRUE`. Otherwise, a detailed comparison is made only if the labels (i.e., rownames) of `target` and `current` are the same (possibly in different orders). In all other cases, a brief description of the differences is returned (sometimes with recommendations to make further comparisons).

This function can be used for testing in programs using `isTRUE` (see examples below).

Value

`TRUE` if the two sets are identical; a list with two elements (message and different.sites) if a detailed comparison is done; or a vector of mode character.

Author(s)

Emmanuel Paradis

See Also

`image.DNAbin`, `clustal`, `checkAlignment`, the generic function: `all.equal`

Examples

```
data(woodmouse)
woodm2 <- woodmouse
woodm2[1, c(1:5, 10:12, 30:40)] <- as.DNAbin("g")
res <- all.equal(woodmouse, woodm2, plot = TRUE)
str(res)

## if used for testing in R programs:
isTRUE(all.equal(woodmouse, woodmouse)) # TRUE
isTRUE(all.equal(woodmouse, woodm2)) # FALSE

all.equal(woodmouse, woodmouse[15:1, ])
all.equal(woodmouse, woodmouse[-1, ])
all.equal(woodmouse, woodmouse[, -1])

## Not run:
## To run the followings you need internet and Clustal and MUSCLE
## correctly installed.
## Data from Johnson et al. (2006, Science)
refs <- paste("DQ082", 505:545, sep = "")
DNA <- read.GenBank(refs)
DNA.clustal <- clustal(DNA)
DNA.muscle <- muscle(DNA)
isTRUE(all.equal(DNA.clustal, DNA.muscle)) # FALSE
all.equal(DNA.clustal, DNA.muscle, TRUE)

## End(Not run)
```

---

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.

Value

A logical value, or a two-column matrix.

Note

The algorithm used here does not work correctly for the comparison of topologies (i.e., ignoring tip labels) of unrooted trees. This also affects unique.multiPhylo which calls the present function. See:

https://www.mail-archive.com/r-sig-phylo@r-project.org/msg01445.html.

Author(s)

Benoît Durand <b.durand@alfort.AFSSA.FR>

See Also

all.equal for the generic R function, comparePhylo

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)
```

`alview` *Print DNA or AA Sequence Alignement*

Description

This function displays in the console or a file an alignment of DNA or AAsequences. The first sequence is printed on the first row and the bases of the other sequences are replaced by dots if they are identical with the first sequence.

Usage

```
alview(x, file = "", uppercase = TRUE, showpos = TRUE)
```

Arguments

`x`	a matrix or a list of DNA sequences (class `"DNAbin"`) or a matrix of AA sequences (class `"AAbin"`).
`file`	a character string giving the name of the file where to print the sequences; by default, they are printed in the console.
`uppercase`	a logical specifying whether to print the bases as uppercase letters.
`showpos`	either a logical value specifying whether to display the site positions, or a numeric vector giving these positions (see examples).

Details

The first line of the output shows the position of the last column of the printed alignment.

Author(s)

Emmanuel Paradis

See Also

`DNAbin`, `image.DNAbin`, `alex`, `clustal`, `checkAlignment`, `all.equal.DNAbin`

Examples

```
data(woodmouse)
alview(woodmouse[, 1:50])
alview(woodmouse[, 1:50], uppercase = FALSE)
## display only some sites:
j <- c(10, 49, 125, 567) # just random
x <- woodmouse[, j]
alview(x, showpos = FALSE) # no site position displayed
alview(x, showpos = j)
## Not run:
alview(woodmouse, file = "woodmouse.txt")

## End(Not run)
```

apetools *Tools to Explore Files*

Description

These functions help to find files on the local disk.

Usage

```
Xplorefiles(from = "HOME", recursive = TRUE, ignore.case = TRUE)
editFileExtensions()
bydir(x)
Xplor(from = "HOME")
```

Arguments

from	the directory where to start the file search; by default, the 'HOME' directory. Use from = getwd() to start from the current working directory.
recursive	whether to search the subdirectories; TRUE by default.
ignore.case	whether to ignore the case of the file extensions; TRUE by default.
x	a list returned by Xplorefiles.

Details

Xplorefiles looks for all files with a specified extension in their names. The default is to look for the following file types: CLUSTAL (.aln), FASTA (.fas, .fasta), FASTQ (.fq, .fastq), NEWICK (.nwk, .newick, .tre, .tree), NEXUS (.nex, .nexus), and PHYLIP (.phy). This list can be modified with editFileExtensions.

bydir sorts the list of files by directories.

Xplor combines the other operations and opens the results in a Web browser with clickable links to the directories and files.

Value

Xplorefiles returns a list. bydir prints the file listings on the console.

Author(s)

Emmanuel Paradis

Examples

```
## Not run:
x <- Xplorefiles()
x # all data files on your disk
bydir(x) # sorted by directories
bydir(x["fasta"]) # only the FASTA files
Xplorefiles(getwd(), recursive = FALSE) # look only in current dir
Xplor()

## End(Not run)
```

`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 'DNAString'
as.DNAbin(x, ...)

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

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

## S3 method for class 'DNAMultipleAlignment'
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. It can convert some S4 classes from the package **Biostrings** in BioConductor. For consistency within **ape**, this uses an S3-style syntax. To convert objects of class "DNAStringSetList", see the examples.

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

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)
### conversion from BioConductor:
## Not run:
if (require(Biostrings)) {
data(phiX174Phage)
X <- as.DNAbin(phiX174Phage)
## base frequencies:
base.freq(X) # from ape
alphabetFrequency(phiX174Phage) # from Biostrings
### for objects of class "DNAStringSetList"
X <- lapply(x, as.DNAbin) # a list of lists
### to put all sequences in a single list:
```

```
X <- unlist(X, recursive = FALSE)
class(X) <- "DNAbin"
}

## End(Not run)
```

`as.bitsplits` *Split Frequencies and Conversion Among Split Classes*

Description

`bitsplits` returns the bipartitions (aka splits) for a single tree or a list of trees. If at least one tree is rooted, an error is returned.

`countBipartitions` returns the frequencies of the bipartitions from a reference tree (phy) observed in a list of trees (X), all unrooted.

`as.bitsplits` and `as.prop.part` are generic functions for converting between the `"bitsplits"` and `"prop.part"` classes.

Usage

```
bitsplits(x)
countBipartitions(phy, X)
as.bitsplits(x)
## S3 method for class 'prop.part'
as.bitsplits(x)
## S3 method for class 'bitsplits'
print(x, ...)
## S3 method for class 'bitsplits'
sort(x, decreasing = FALSE, ...)
as.prop.part(x, ...)
## S3 method for class 'bitsplits'
as.prop.part(x, include.trivial = FALSE, ...)
```

Arguments

`x`	an object of the appropriate class.
`phy`	an object of class `"phylo"`.
`X`	an object of class `"multiPhylo"`.
`decreasing`	a logical value to sort the bipartitions in increasing (the default) or decreasing order of their frequency.
`include.trivial`	a logical value specifying whether to include the trivial split with all tips in the returned object.
`...`	further arguments passed to or from other methods.

Details

These functions count bipartitions as defined by internal branches, so they work only with unrooted trees. The structure of the class "bitsplits" is described in a separate document on ape's web site.

This data structure has a memory requirement proportional to n^2, so it can be inefficient with large trees (> 1000 tips), particularly if they are very different (i.e., with few shared splits). In any case, an error occurs if the product of the number of tips by the number of nodes is greater than $2^{31} - 1$ (~2.1 billion). A warning message is given if the tree(s) has(ve) more than 46,341 tips. It may happen that the search for splits is interrupted if the data structure is full (with a warning message).

Value

bitsplits, as.bitsplits, and sort return an object of class "bitsplits".

countBipartitions returns a vector of integers.

as.prop.part returns an object of class "prop.part".

Author(s)

Emmanuel Paradis

See Also

prop.part, is.compatible

Examples

```
tr <- rtree(20)
pp <- prop.part(tr)
as.bitsplits(pp)
## works only with unrooted trees (ape 5.5):
countBipartitions(rtree(10, rooted = FALSE), rmtree(100, 10, rooted = FALSE))
```

---

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 two-column numeric matrix where the columns represent the sibling pairs.
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

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 and Network Objects*

Description

as.phylo is a generic function which converts an object into a tree of class "phylo". There are currently two methods for objects of class "hclust" and of class "phylog" (implemented in the package **ade4**). The default method is for any object inheriting the class "phylo" which is returned unchanged.

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).

as.network and as.igraph convert trees of class "phylo" into these respective classes defined in the packages of the same names (where the generics are defined).

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

Usage

```
as.phylo(x, ...)
## Default S3 method:
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)
## S3 method for class 'phylo'
as.network(x, directed = is.rooted(x), ...)
## S3 method for class 'phylo'
as.igraph(x, directed = is.rooted(x), use.labels = TRUE, ...)
```

Arguments

x	an object to be converted into another class.
directed	a logical value: should the network be directed? By default, this depends on whether the tree is rooted or not.
use.labels	a logical specifying whether to use labels to build the network of class "igraph". If TRUE and the tree has no node labels, then some default labels are created first. If FALSE, the network is built with integers.
...	further arguments to be passed to or from other methods.
phy	an object of class "phylo".

Value

An object of class "hclust", "phylo", "network", or "igraph".

Note

In an object of class "hclust", the height gives the distance between the two sets that are being agglomerated. So these distances are divided by two when setting the branch lengths of a phylogenetic tree.

Author(s)

Emmanuel Paradis

See Also

hclust, as.hclust, dendrogram, as.phylo.formula

Examples

```
data(bird.orders)
hc <- as.hclust(bird.orders)
tr <- as.phylo(hc)
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 identical plots with
### plot.phylo and plot.dendrogram:
layout(matrix(1:2, 2, 1))
plot(bird.orders, font = 1, no.margin = TRUE, label.offset = 0.4)
par(mar = c(0, 0, 0, 8))
plot(dend, horiz = TRUE)
```

```
layout(1)

## Not run:
### convert into networks:
if (require(network)) {
    x <- as.network(rtree(10))
    print(x)
    plot(x, vertex.cex = 1:4)
    plot(x, displaylabels = TRUE)
}
tr <- rtree(5)
if (require(igraph)) {
    print((x <- as.igraph(tr)))
    plot(x)
    print(as.igraph(tr, TRUE, FALSE))
    print(as.igraph(tr, FALSE, FALSE))
}

## End(Not run)
```

`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(), collapse = TRUE, ...)
```

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 user's workspace).
`collapse`	a logical value specifying whether to collapse single nodes in the returned tree (see details).
`...`	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 will contain polytomies.

The option `collapse = FALSE` has for effect to add single nodes in the tree when a given higher level has only one element in the level below (e.g., a monospecific genus); see the example below.

Value

an object of class "phylo".

Author(s)

Julien Dutheil <dutheil@evolbio.mpg.de>, Eric Marcon and Klaus Schliep

See Also

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

Examples

```
data(carnivora)
frm <- ~SuperFamily/Family/Genus/Species
tr <- as.phylo(frm, data = carnivora, collapse=FALSE)
tr$edge.length <- rep(1, nrow(tr$edge))
plot(tr, show.node.label=TRUE)
Nnode(tr)
## compare with:
Nnode(as.phylo(frm, data = carnivora, collapse = FALSE))
```

---

axisPhylo *Axis on Side of Phylogeny*

Description

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

Usage

```
axisPhylo(side = NULL, root.time = NULL, backward = TRUE, ...)
```

Arguments

side	a numeric value specifying the side where the axis is plotted: 1: below, 2: left, 3: above, 4: right. By default, this is taken from the direction of the plot.
root.time	the time assigned to the root node of the tree. By default, this is taken from the root.time element of the tree. If it is absent, this is determined from the next option.
backward	a logical value; if TRUE, the most distant tip from the root is considered as the origin of the time scale; if FALSE, this is the root node.
...	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, Klaus Schliep

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(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

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

base.freq computes the frequencies (absolute or relative) of the four DNA bases (adenine, cytosine, guanine, and thymidine) from a sample of sequences.

GC.content computes the proportion of G+C (using the previous function). All missing or unknown sites are ignored.

Ftab computes the contingency table with the absolute frequencies of the DNA bases from a pair of sequences.

Usage

```
base.freq(x, freq = FALSE, all = FALSE)
GC.content(x)
Ftab(x, y = NULL)
```

Arguments

x	a vector, a matrix, or a list which contains the DNA sequences.
y	a vector with a single DNA sequence.
freq	a logical specifying whether to return the proportions (the default) or the absolute frequencies (counts).
all	a logical; by default only the counts of A, C, G, and T are returned. If all = TRUE, all counts of bases, ambiguous codes, missing data, and alignment gaps are returned.

Details

The base frequencies are computed over all sequences in the sample.

For Ftab, if the argument y is given then both x and y are coerced as vectors and must be of equal length. If y is not given, x must be a matrix or a list and only the two first sequences are used.

Value

A numeric vector with names c("a", "c", "g", "t") (and possibly "r", "m", ...), a single numeric value, or a four by four matrix with similar dimnames.

Author(s)

Emmanuel Paradis

See Also

seg.sites, nuc.div (in **pegas**), DNAbin

Examples

```
data(woodmouse)
base.freq(woodmouse)
base.freq(woodmouse, TRUE)
base.freq(woodmouse, TRUE, TRUE)
GC.content(woodmouse)
Ftab(woodmouse)
Ftab(woodmouse[1, ], woodmouse[2, ]) # same than above
Ftab(woodmouse[14:15, ]) # between the last two
```

`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, conditional = TRUE)
```

Arguments

`phy`	an object of class `"phylo"`.
`S`	a numeric vector giving the number of species for each tip.
`conditional`	whether probabilities should be conditioned on no extinction (mainly to compare results with previous analyses; see details).

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.

If `conditional = TRUE`, the probabilities of the taxonomic data are calculated conditioned on no extinction (Rabosky et al. 2007). In previous versions of the present function (until ape 2.6-1), unconditional probabilities were used resulting in underestimated extinction rate. Though it does

not make much sense to use `conditional = FALSE`, this option is provided to compare results from previous analyses: if the species richnesses are relatively low, both versions will give similar results (see examples).

Author(s)

Emmanuel Paradis

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.

Rabosky, D. L., Donnellan, S. C., Talaba, A. L. and Lovette, I. J. (2007) Exceptional among-lineage variation in diversification rates during the radiation of Australia's most diverse vertebrate clade. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **274**, 2915–2923.

See Also

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

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)
bd.ext(bird.orders, S, FALSE) # same than older versions
```

---

bd.time *Time-Dependent Birth-Death Models*

Description

This function fits a used-defined time-dependent birth-death model.

Usage

```
bd.time(phy, birth, death, BIRTH = NULL, DEATH = NULL,
        ip, lower, upper, fast = FALSE, boot = 0, trace = 0)
```

Arguments

phy	an object of class "phylo".
birth	either a numeric (if speciation rate is assumed constant), or a (vectorized) function specifying how the birth (speciation) probability changes through time (see details).
death	id. for extinction probability.
BIRTH	(optional) a vectorized function giving the primitive of birth.
DEATH	id. for death.
ip	a numeric vector used as initial values for the estimation procedure. If missing, these values are guessed.
lower, upper	the lower and upper bounds of the parameters. If missing, these values are guessed too.
fast	a logical value specifying whether to use faster integration (see details).
boot	the number of bootstrap replicates to assess the confidence intervals of the parameters. Not run by default.
trace	an integer value. If non-zero, the fitting procedure is printed every trace steps. This can be helpful if convergence is particularly slow.

Details

Details on how to specify the birth and death functions and their primitives can be found in the help page of yule.time.

The model is fitted by minimizing the least squares deviation between the observed and the predicted distributions of branching times. These computations rely heavily on numerical integrations. If fast = FALSE, integrations are done with R's integrate function. If fast = TRUE, a faster but less accurate function provided in **ape** is used. If fitting a complex model to a large phylogeny, a strategy might be to first use the latter option, and then to use the estimates as starting values with fast = FALSE.

Value

A list with the following components:

- par: a vector of estimates with names taken from the parameters in the specified functions.
- SS: the minimized sum of squares.
- convergence: output convergence criterion from nlminb.
- message: id.
- iterations: id.
- evaluations: id.

Author(s)

Emmanuel Paradis

References

Paradis, E. (2011) Time-dependent speciation and extinction from phylogenies: a least squares approach. *Evolution*, **65**, 661–672.

See Also

ltt.plot, birthdeath, yule.time, LTT

Examples

```
set.seed(3)
tr <- rbdtree(0.1, 0.02)
bd.time(tr, 0, 0) # fits a simple BD model
bd.time(tr, 0, 0, ip = c(.1, .01)) # 'ip' is useful here
## the classic logistic:
birth.logis <- function(a, b) 1/(1 + exp(-a*t - b))
## Not run:
bd.time(tr, birth.logis, 0, ip = c(0, -2, 0.01))
## slow to get:
## $par
##            a            b        death
## -0.003486961 -1.995983179  0.016496454
##
## $SS
## [1] 20.73023

## End(Not run)
```

---

binaryPGLMM *Phylogenetic Generalized Linear Mixed Model for Binary Data*

Description

binaryPGLMM performs linear regression for binary phylogenetic data, estimating regression coefficients with approximate standard errors. It simultaneously estimates the strength of phylogenetic signal in the residuals and gives an approximate conditional likelihood ratio test for the hypothesis that there is no signal. Therefore, when applied without predictor (independent) variables, it gives a test for phylogenetic signal for binary data. The method uses a GLMM approach, alternating between penalized quasi-likelihood (PQL) to estimate the "mean components" and restricted maximum likelihood (REML) to estimate the "variance components" of the model.

binaryPGLMM.sim is a companion function that simulates binary phylogenetic data of the same structure analyzed by binaryPGLMM.

Usage

```
binaryPGLMM(formula, data = list(), phy, s2.init = 0.1,
            B.init = NULL, tol.pql = 10^-6, maxit.pql = 200,
            maxit.reml = 100)
```

```
binaryPGLMM.sim(formula, data = list(), phy, s2 = NULL, B = NULL, nrep = 1)

## S3 method for class 'binaryPGLMM'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

formula	a two-sided linear formula object describing the fixed-effects of the model; for example, Y ~ X.
data	a data frame containing the variables named in formula.
phy	a phylogenetic tree as an object of class "phylo".
s2.init	an initial estimate of s2, the scaling component of the variance in the PGLMM. A value of s2 = 0 implies no phylogenetic signal. Note that the variance-covariance matrix given by the phylogeny phy is scaled to have determinant = 1.
B.init	initial estimates of B, the matrix containing regression coefficients in the model. This matrix must have dim(B.init)=c(p+1,1), where p is the number of predictor (independent) variables; the first element of B corresponds to the intercept, and the remaining elements correspond in order to the predictor (independent) variables in the model.
tol.pql	a control parameter dictating the tolerance for convergence for the PQL optimization.
maxit.pql	a control parameter dictating the maximum number of iterations for the PQL optimization.
maxit.reml	a control parameter dictating the maximum number of iterations for the REML optimization.
x	an object of class "binaryPGLMM".
s2	in binaryPGLMM.sim, value of s2. See s2.init.
B	in binaryPGLMM.sim, value of B, the matrix containing regression coefficients in the model. See B.init.
nrep	in binaryPGLMM.sim, number of compete data sets produced.
digits	the number of digits to print.
...	further arguments passed to print.

Details

The function estimates parameters for the model

$$Pr(Y = 1) = q$$

$$q = inverse.logit(b0 + b1 * x1 + b2 * x2 + \cdots + \epsilon)$$

$$\epsilon\ Gaussian(0, s2 * V)$$

where V is a variance-covariance matrix derived from a phylogeny (typically under the assumption of Brownian motion evolution). Although mathematically there is no requirement for V to be ultrametric, forcing V into ultrametric form can aide in the interpretation of the model, because in regression for binary dependent variables, only the off-diagonal elements (i.e., covariances) of matrix V are biologically meaningful (see Ives & Garland 2014).

The function converts a phylo tree object into a variance-covariance matrix, and further standardizes this matrix to have determinant = 1. This in effect standardizes the interpretation of the scalar s2. Although mathematically not required, it is a very good idea to standardize the predictor (independent) variables to have mean 0 and variance 1. This will make the function more robust and improve the interpretation of the regression coefficients. For categorical (factor) predictor variables, you will need to construct 0-1 dummy variables, and these should not be standardized (for obvious reasons).

The estimation method alternates between PQL to obtain estimates of the mean components of the model (this is the standard approach to estimating GLMs) and REML to obtain estimates of the variance components. This method gives relatively fast and robust estimation. Nonetheless, the estimates of the coefficients B will generally be upwards bias, as is typical of estimation for binary data. The standard errors of B are computed from the PQL results conditional on the estimate of s2 and therefore should tend to be too small. The function returns an approximate P-value for the hypothesis of no phylogenetic signal in the residuals (i.e., H0:s2 = 0) using an approximate likelihood ratio test based on the conditional REML likelihood (rather than the marginal likelihood). Simulations have shown that these P-values tend to be high (giving type II errors: failing to identify variances that in fact are statistically significantly different from zero).

It is a good idea to confirm statistical inferences using parametric bootstrapping, and the companion function binaryPGLMM.sim gives a simply tool for this. See Examples below.

Value

An object of class "binaryPGLMM".

`formula`	formula specifying the regression model.
`B`	estimates of the regression coefficients.
`B.se`	approximate PQL standard errors of the regression coefficients.
`B.cov`	approximate PQL covariance matrix for the regression coefficients.
`B.zscore`	approximate PQL Z scores for the regression coefficients.
`B.pvalue`	approximate PQL tests for the regression coefficients being different from zero.
`s2`	phylogenetic signal measured as the scalar magnitude of the phylogenetic variance-covariance matrix s2 * V.
`P.H0.s2`	approximate likelihood ratio test of the hypothesis H0 that s2 = 0. This test is based on the conditional REML (keeping the regression coefficients fixed) and is prone to inflated type 1 errors.
`mu`	for each data point y, the estimate of p that y = 1.
`b`	for each data point y, the estimate of inverse.logit(p).
`X`	the predictor (independent) variables returned in matrix form (including 1s in the first column).
`H`	residuals of the form b + (Y - mu)/(mu * (1 - mu)).

B.init	the user-provided initial estimates of B. If B.init is not provided, these are estimated using glm() assuming no phylogenetic signal. The glm() estimates can generate convergence problems, so using small values (e.g., 0.01) is more robust but slower.
VCV	the standardized phylogenetic variance-covariance matrix.
V	estimate of the covariance matrix of H.
convergeflag	flag for cases when convergence failed.
iteration	number of total iterations performed.
converge.test.B	final tolerance for B.
converge.test.s2	final tolerance for s2.
rcondflag	number of times B is reset to 0.01. This is done when rcond(V) < 10^(-10), which implies that V cannot be inverted.
Y	in binaryPGLMM.sim, the simulated values of Y.

Author(s)

Anthony R. Ives

References

Ives, A. R. and Helmus, M. R. (2011) Generalized linear mixed models for phylogenetic analyses of community structure. *Ecological Monographs*, **81**, 511–525.

Ives, A. R. and Garland, T., Jr. (2014) Phylogenetic regression for binary dependent variables. Pages 231–261 *in* L. Z. Garamszegi, editor. *Modern Phylogenetic Comparative Methods and Their Application in Evolutionary Biology*. Springer-Verlag, Berlin Heidelberg.

See Also

package **pez** and its function communityPGLMM; package **phylolm** and its function phyloglm; package **MCMCglmm**

Examples

```
## Illustration of binaryPGLMM() with simulated data

# Generate random phylogeny

n <- 100
phy <- compute.brlen(rtree(n=n), method = "Grafen", power = 1)

# Generate random data and standardize to have mean 0 and variance 1
X1 <- rTraitCont(phy, model = "BM", sigma = 1)
X1 <- (X1 - mean(X1))/var(X1)

# Simulate binary Y
sim.dat <- data.frame(Y=array(0, dim=n), X1=X1, row.names=phy$tip.label)
```

```
sim.dat$Y <- binaryPGLMM.sim(Y ~ X1, phy=phy, data=sim.dat, s2=.5,
                             B=matrix(c(0,.25),nrow=2,ncol=1), nrep=1)$Y

# Fit model
binaryPGLMM(Y ~ X1, phy=phy, data=sim.dat)

## Not run:
# Compare with phyloglm()
library(phylolm)
summary(phyloglm(Y ~ X1, phy=phy, data=sim.dat))

# Compare with glm() that does not account for phylogeny
summary(glm(Y ~ X1, data=sim.dat, family="binomial"))

# Compare with logistf() that does not account
# for phylogeny but is less biased than glm()
library(logistf)
logistf(Y ~ X1, data=sim.dat)

# Compare with MCMCglmm
library(MCMCglmm)

V <- vcv(phy)
V <- V/max(V)
detV <- exp(determinant(V)$modulus[1])
V <- V/detV^(1/n)

invV <- Matrix(solve(V),sparse=T)
sim.dat$species <- phy$tip.label
rownames(invV) <- sim.dat$species

nitt <- 43000
thin <- 10
burnin <- 3000

prior <- list(R=list(V=1, fix=1), G=list(G1=list(V=1, nu=1000, alpha.mu=0, alpha.V=1)))
summary(MCMCglmm(Y ~ X1, random=~species, ginvers=list(species=invV),
    data=sim.dat, slice=TRUE, nitt=nitt, thin=thin, burnin=burnin,
    family="categorical", prior=prior, verbose=FALSE))

## Examine bias in estimates of B1 and s2 from binaryPGLMM with
# simulated data. Note that this will take a while.

Reps = 1000

s2 <- 0.4
B1 <- 1

meanEsts <- data.frame(n = Inf, B1 = B1, s2 = s2, Pr.s2 = 1, propconverged = 1)

for (n in c(160, 80, 40, 20)) {

  meanEsts.n <- data.frame(B1 = 0, s2 = 0, Pr.s2 = 0, convergefailure = 0)
```

```
    for (rep in 1:Reps) {
      phy <- compute.brlen(rtree(n = n), method = "Grafen", power = 1)
      X <- rTraitCont(phy, model = "BM", sigma = 1)
      X <- (X - mean(X))/var(X)

      sim.dat <- data.frame(Y = array(0, dim = n), X = X, row.names = phy$tip.label)
      sim <- binaryPGLMM.sim(Y ~ 1 + X, phy = phy, data = sim.dat, s2 = s2,
                                      B = matrix(c(0,B1), nrow = 2, ncol = 1), nrep = 1)
      sim.dat$Y <- sim$Y

      z <- binaryPGLMM(Y ~ 1 + X, phy = phy, data = sim.dat)

      meanEsts.n[rep, ] <- c(z$B[2], z$s2, z$P.H0.s2, z$convergeflag == "converged")
  }
converged <- meanEsts.n[,4]
meanEsts <- rbind(meanEsts,
                  c(n, mean(meanEsts.n[converged==1,1]),
                           mean(meanEsts.n[converged==1,2]),
                           mean(meanEsts.n[converged==1, 3] < 0.05),
                           mean(converged)))
}
meanEsts

# Results output for B1 = 0.5, s2 = 0.4; n-Inf gives the values used to
# simulate the data
#    n       B1        s2      Pr.s2 propconverged
# 1 Inf 1.000000 0.4000000 1.00000000         1.000
# 2 160 1.012719 0.4479946 0.36153072         0.993
# 3  80 1.030876 0.5992027 0.24623116         0.995
# 4  40 1.110201 0.7425203 0.13373860         0.987
# 5  20 1.249886 0.8774708 0.05727377         0.873


## Examine type I errors for estimates of B0 and s2 from binaryPGLMM()
# with simulated data. Note that this will take a while.

Reps = 1000

s2 <- 0
B0 <- 0
B1 <- 0

H0.tests <- data.frame(n = Inf, B0 = B0, s2 = s2, Pr.B0 = .05,
                       Pr.s2 = .05, propconverged = 1)
for (n in c(160, 80, 40, 20)) {

  ests.n <- data.frame(B1 = 0, s2 = 0, Pr.B0 = 0, Pr.s2 = 0, convergefailure = 0)
  for (rep in 1:Reps) {
    phy <- compute.brlen(rtree(n = n), method = "Grafen", power = 1)
    X <- rTraitCont(phy, model = "BM", sigma = 1)
    X <- (X - mean(X))/var(X)

    sim.dat <- data.frame(Y = array(0, dim = n), X = X, row.names = phy$tip.label)
```

```
    sim <- binaryPGLMM.sim(Y ~ 1, phy = phy, data = sim.dat, s2 = s2,
                           B = matrix(B0, nrow = 1, ncol = 1), nrep = 1)
    sim.dat$Y <- sim$Y

    z <- binaryPGLMM(Y ~ 1, phy = phy, data = sim.dat)

   ests.n[rep, ] <- c(z$B[1], z$s2, z$B.pvalue, z$P.H0.s2, z$convergeflag == "converged")
  }

converged <- ests.n[,5]
H0.tests <- rbind(H0.tests,
                  c(n, mean(ests.n[converged==1,1]),
                    mean(ests.n[converged==1,2]),
                    mean(ests.n[converged==1, 3] < 0.05),
                    mean(ests.n[converged==1, 4] < 0.05),
                    mean(converged)))
}
H0.tests

# Results for type I errors for B0 = 0 and s2 = 0; n-Inf gives the values
# used to simulate the data. These results show that binaryPGLMM() tends to
# have lower-than-nominal p-values; fewer than 0.05 of the simulated
# data sets have H0:B0=0 and H0:s2=0 rejected at the alpha=0.05 level.
#     n            B0         s2      Pr.B0      Pr.s2 propconverged
# 1 Inf  0.0000000000 0.00000000 0.05000000 0.05000000         1.000
# 2 160 -0.0009350357 0.07273163 0.02802803 0.04804805         0.999
# 3  80 -0.0085831477 0.12205876 0.04004004 0.03403403         0.999
# 4  40  0.0019303847 0.25486307 0.02206620 0.03711133         0.997
# 5  20  0.0181394905 0.45949266 0.02811245 0.03313253         0.996

## End(Not run)
```

`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, interactive = FALSE)
x + y
```

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.
interactive	if TRUE the user is asked to choose the tip or node of x by clicking on the tree which must be plotted.

Details

The argument x can be seen as the receptor tree, whereas y is the donor tree. The root of y is then grafted 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.

x + y is a shortcut for:

```
bind.tree(x, y, position = if (is.null(x$root.edge)) 0 else
x$root.edge)
```

If only one of the trees has no branch length, the branch lengths of the other one are ignored with a warning.

If one (or both) of the trees has no branch length, it is possible to specify a value of 'position' to graft 'y' below the node of 'x' specified by 'where'. In this case, the exact value of 'position' is not important as long as it is greater than zero. The new node will be multichotomous if 'y' has no root edge. This can be solved by giving an arbitrary root edge to 'y' beforehand (e.g., y$root.edge <- 1): it will be deleted during the binding operation.

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis

See Also

drop.tip, root

Examples

```
### binds the two clades of bird orders
treefile1 <- tempfile("tree", fileext = ".tre")
treefile2 <- tempfile("tree", fileext = ".tre")
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 = treefile1, 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 = treefile2, sep = "\n")
tree.bird1 <- read.tree(treefile1)
tree.bird2 <- read.tree(treefile2)
unlink(c(treefile1, treefile2)) # clean-up
(birds <- tree.bird1 + tree.bird2)
layout(matrix(c(1, 2, 3, 3), 2, 2))
plot(tree.bird1)
plot(tree.bird2)
plot(birds)

### examples with random trees
x <- rtree(4, tip.label = LETTERS[1:4])
y <- rtree(4, tip.label = LETTERS[5:8])
x <- makeNodeLabel(x, prefix = "x_")
y <- makeNodeLabel(y, prefix = "y_")
x$root.edge <- y$root.edge <- .2

z <- bind.tree(x, y, po=.2)
plot(y, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("y")
plot(x, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("x")
plot(z, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("z <- bind.tree(x, y, po=.2)")

## make sure the terminal branch length is long enough:
x$edge.length[x$edge[, 2] == 2] <- 0.2

z <- bind.tree(x, y, 2, .1)
plot(y, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("y")
plot(x, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("x")
plot(z, show.node.label = TRUE, font = 1, root.edge = TRUE)
title("z <- bind.tree(x, y, 2, .1)")

x <- rtree(50)
y <- rtree(50)
x$root.edge <- y$root.edge <- .2
z <- x + y
plot(y, show.tip.label = FALSE, root.edge = TRUE); axisPhylo()
title("y")
plot(x, show.tip.label = FALSE, root.edge = TRUE); axisPhylo()
title("x")
plot(z, show.tip.label = FALSE, root.edge = TRUE); axisPhylo()
title("z <- x + y")
layout(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, mvr, bionjs, SDM, dist.dna

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[lower.tri(M)] <- x
M <- t(M)
M[lower.tri(M)] <- x
dimnames(M) <- list(1:8, 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(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

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, bd.time

---

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 .... If a single tree is passed, the returned object is a list of vectors with the tips descending from each node (i.e., clade compositions indexed by node number).

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, quiet = FALSE,
           rooted = is.rooted(phy), jumble = TRUE,
            mc.cores = 1)
prop.part(..., check.labels = TRUE)
prop.clades(phy, ..., part = NULL, rooted = FALSE)
## 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, col = "red", ...)
```

Arguments

phy	an object of class "phylo".
x	in the case of boot.phylo: a taxa (rows) by characters (columns) matrix; 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).
quiet	a logical: a progress bar is displayed by default.
rooted	a logical specifying whether the trees should be treated as rooted or not.
jumble	a logical value. By default, the rows of x are randomized to avoid artificially too large bootstrap values associated with very short branches.

mc.cores	the number of cores (CPUs) to be used (passed to **parallel**).
...	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, 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.
col	the colour used to visualise the bipartitions.

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 check.labels = FALSE in prop.part decreases 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).

Since **ape** 3.5, prop.clades should return sensible results for all values of rooted: if FALSE, the numbers of bipartitions (or splits); if TRUE, the number of clades (of hopefully rooted trees).

Author(s)

Emmanuel Paradis

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

as.bitsplits, dist.topo, consensus, nodelabels

Examples

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

set.seed(2)
tr <- rtree(10) # rooted
## the following used to return a wrong result with ape <= 3.4:
prop.clades(tr, tr)
prop.clades(tr, tr, rooted = TRUE)
tr <- rtree(10, rooted = FALSE)
prop.clades(tr, tr) # correct

### an illustration of the use of prop.clades with bootstrap trees:
```

```
fun <- function(x) as.phylo(hclust(dist.dna(x), "average")) # upgma() in phangorn
tree <- fun(woodmouse)
## get 100 bootstrap trees:
bstrees <- boot.phylo(tree, woodmouse, fun, trees = TRUE)$trees
## get proportions of each clade:
clad <- prop.clades(tree, bstrees, rooted = TRUE)
## get proportions of each bipartition:
boot <- prop.clades(tree, bstrees)
layout(1)
par(mar = rep(2, 4))
plot(tree, main = "Bipartition vs. Clade Support Values")
drawSupportOnEdges(boot)
nodelabels(clad)
legend("bottomleft", legend = c("Bipartitions", "Clades"), pch = 22,
       pt.bg = c("green", "lightblue"), pt.cex = 2.5)

## Not run:
## an example of double bootstrap:
nrep1 <- 100
nrep2 <- 100
p <- ncol(woodmouse)
DB <- 0

for (b in 1:nrep1) {
    X <- woodmouse[, sample(p, p, TRUE)]
    DB <- DB + boot.phylo(tr, X, f, nrep2, quiet = TRUE)
}
DB
## to compare with:
boot.phylo(tr, woodmouse, f, 1e4)

## End(Not run)
```

`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

See Also

is.ultrametric

---

c.phylo *Building Lists of Trees*

Description

These functions help to build lists of trees of class "multiPhylo".

Usage

```
## S3 method for class 'phylo'
c(..., recursive = TRUE)
## S3 method for class 'multiPhylo'
c(..., recursive = TRUE)
.compressTipLabel(x, ref = NULL)
.uncompressTipLabel(x)
```

Arguments

...	one or several objects of class "phylo" and/or "multiPhylo".
recursive	see details.
x	an object of class "phylo" or "multiPhylo".
ref	an optional vector of mode character to constrain the order of the tips. By default, the order from the first tree is used.

Details

These c methods check all the arguments, and return by default a list of single trees unless some objects are not trees or lists of trees, in which case recursive is switched to FALSE and a warning message is given. If recursive = FALSE, the objects are simply concatenated into a list. Before **ape** 4.0, recursive was always set to FALSE.

.compressTipLabel transforms an object of class "multiPhylo" by checking that all trees have the same tip labels and renumbering the tips in the edge matrix so that the tip numbers are also the same taking the first tree as the reference (duplicated labels are not allowed). The returned object has a unique vector of tip labels (attr(x, "TipLabel")).

.uncompressTipLabel does the reverse operation.

Value

An object of class "multiPhylo".

Author(s)

Emmanuel Paradis

See Also

summary.phylo, multiphylo

Examples

```
x <- c(rtree(4), rtree(2))
x
y <- c(rtree(4), rtree(4))
z <- c(x, y)
z
print(z, TRUE)
try(.compressTipLabel(x)) # error
a <- .compressTipLabel(y)
.uncompressTipLabel(a) # back to y
## eventually compare str(a) and str(y)
```

CADM.global *Congruence among distance matrices*

Description

Function CADM.global compute and test the coefficient of concordance among several distance matrices through a permutation test.

Function CADM.post carries out a posteriori permutation tests of the contributions of individual distance matrices to the overall concordance of the group.

Use in phylogenetic analysis: to identify congruence among distance matrices (D) representing different genes or different types of data. Congruent D matrices correspond to data tables that can be used together in a combined phylogenetic or other type of multivariate analysis.

Usage

```
CADM.global(Dmat, nmat, n, nperm=99, make.sym=TRUE, weights=NULL,
            silent=FALSE)
CADM.post  (Dmat, nmat, n, nperm=99, make.sym=TRUE, weights=NULL,
             mult="holm", mantel=FALSE, silent=FALSE)
```

Arguments

Dmat	A text file listing the distance matrices one after the other, with or without blank lines in-between. Each matrix is in the form of a square distance matrix with 0's on the diagonal.
nmat	Number of distance matrices in file Dmat.
n	Number of objects in each distance matrix. All matrices must have the same number of objects.
nperm	Number of permutations for the tests of significance.
make.sym	TRUE: turn asymmetric matrices into symmetric matrices by averaging the two triangular portions. FALSE: analyse asymmetric matrices as they are.
weights	A vector of positive weights for the distance matrices. Example: weights = c(1,2,3). NULL (default): all matrices have same weight in the calculation of W.
mult	Method for correcting P-values in multiple testing. The methods are "holm" (default), "sidak", and "bonferroni". The Bonferroni correction is overly conservative; it is not recommended. It is included to allow comparisons with the other methods.
mantel	TRUE: Mantel statistics will be computed from ranked distances, as well as permutational P-values. FALSE (default): Mantel statistics and tests will not be computed.
silent	TRUE: informative messages will not be printed, but stopping messages will. Option useful for simulation work. FALSE: informative messages will be printed.

Details

Dmat must contain two or more distance matrices, listed one after the other, all of the same size, and corresponding to the same objects in the same order. Raw data tables can be transformed into distance matrices before comparison with other such distance matrices, or with data that have been obtained as distance matrices, e.g. serological or DNA hybridization data. The distances will be transformed to ranks before computation of the coefficient of concordance and other statistics.

CADM.global tests the global null hypothesis that all matrices are incongruent. If the global null is rejected, function CADM.post can be used to identify the concordant (H0 rejected) and discordant matrices (H0 not rejected) in the group. If a distance matrix has a negative value for the Mantel.mean statistic, that matrix clearly does not belong to the group. Remove that matrix (if there are more than one, remove first the matrix that has the most strongly negative value for Mantel.mean) and run the analysis again.

The corrections used for multiple testing are applied to the list of P-values (P) produced in the a posteriori tests; they take into account the number of tests (k) carried out simulatenously (number of matrices, parameter nmat).

The Holm correction is computed after ordering the P-values in a list with the smallest value to the left. Compute adjusted P-values as:

$$P_{corr} = (k - i + 1) * P$$

where i is the position in the ordered list. Final step: from left to right, if an adjusted P_{corr} in the ordered list is smaller than the one occurring at its left, make the smallest one equal to the largest one.

The Sidak correction is:

$$P_{corr} = 1 - (1 - P)^k$$

The Bonferonni correction is:

$$P_{corr} = k * P$$

Value

`CADM.global` produces a small table containing the W, Chi2, and Prob.perm statistics described in the following list. `CADM.post` produces a table stored in element `A_posteriori_tests`, containing Mantel.mean, Prob, and Corrected.prob statistics in rows; the columns correspond to the k distance matrices under study, labeled Dmat.1 to Dmat.k. If parameter `mantel` is TRUE, tables of Mantel statistics and P-values are computed among the matrices.

`W`	Kendall's coefficient of concordance, W (Kendall and Babington Smith 1939; see also Legendre 2010).
`Chi2`	Friedman's chi-square statistic (Friedman 1937) used in the permutation test of W.
`Prob.perm`	Permutational probability.
`Mantel.mean`	Mean of the Mantel correlations, computed on rank-transformed distances, between the distance matrix under test and all the other matrices in the study.
`Prob`	Permutational probabilities, uncorrected.
`Corrected prob`	Permutational probabilities corrected using the method selected in parameter `mult`.
`Mantel.cor`	Matrix of Mantel correlations, computed on rank-transformed distances, among the distance matrices.
`Mantel.prob`	One-tailed P-values associated with the Mantel correlations of the previous table. The probabilities are computed in the right-hand tail. H0 is tested against the alternative one-tailed hypothesis that the Mantel correlation under test is positive. No correction is made for multiple testing.

Author(s)

Pierre Legendre, Universite de Montreal

References

Campbell, V., Legendre, P. and Lapointe, F.-J. (2009) Assessing congruence among ultrametric distance matrices. *Journal of Classification*, **26**, 103–117.

Campbell, V., Legendre, P. and Lapointe, F.-J. (2011) The performance of the Congruence Among Distance Matrices (CADM) test in phylogenetic analysis. *BMC Evolutionary Biology*, **11**, 64.

Friedman, M. (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, **32**, 675–701.

Kendall, M. G. and Babington Smith, B. (1939) The problem of m rankings. *Annals of Mathematical Statistics*, **10**, 275–287.

Lapointe, F.-J., Kirsch, J. A. W. and Hutcheon, J. M. (1999) Total evidence, consensus, and bat phylogeny: a distance-based approach. *Molecular Phylogenetics and Evolution*, **11**, 55–66.

Legendre, P. (2010) Coefficient of concordance. Pp. 164-169 in: Encyclopedia of Research Design, Vol. 1. N. J. Salkind, ed. SAGE Publications, Inc., Los Angeles.

Legendre, P. and Lapointe, F.-J. (2004) Assessing congruence among distance matrices: single malt Scotch whiskies revisited. *Australian and New Zealand Journal of Statistics*, **46**, 615–629.

Legendre, P. and Lapointe, F.-J. (2005) Congruence entre matrices de distance. P. 178-181 in: Makarenkov, V., G. Cucumel et F.-J. Lapointe [eds] Comptes rendus des 12emes Rencontres de la Societe Francophone de Classification, Montreal, 30 mai - 1er juin 2005.

Siegel, S. and Castellan, N. J., Jr. (1988) *Nonparametric statistics for the behavioral sciences. 2nd edition*. New York: McGraw-Hill.

Examples

```
# Examples 1 and 2: 5 genetic distance matrices computed from simulated DNA
# sequences representing 50 taxa having evolved along additive trees with
# identical evolutionary parameters (GTR+ Gamma + I). Distance matrices were
# computed from the DNA sequence matrices using a p distance corrected with the
# same parameters as those used to simulate the DNA sequences. See Campbell et
# al. (2009) for details.

# Example 1: five independent additive trees. Data provided by V. Campbell.

data(mat5Mrand)
res.global <- CADM.global(mat5Mrand, 5, 50)

# Example 2: three partly similar trees, two independent trees.
# Data provided by V. Campbell.

data(mat5M3ID)
res.global <- CADM.global(mat5M3ID, 5, 50)
res.post   <- CADM.post(mat5M3ID, 5, 50, mantel=TRUE)

# Example 3: three matrices respectively representing Serological
# (asymmetric), DNA hybridization (asymmetric) and Anatomical (symmetric)
# distances among 9 families. Data from Lapointe et al. (1999).

data(mat3)
res.global <- CADM.global(mat3, 3, 9, nperm=999)
res.post   <- CADM.post(mat3, 3, 9, nperm=999, mantel=TRUE)

# Example 4, showing how to bind two D matrices (cophenetic matrices
# in this example) into a file using rbind(), then run the global test.

a <- rtree(5)
```

```
b <- rtree(5)
A <- cophenetic(a)
B <- cophenetic(b)
x <- rownames(A)
B <- B[x, x]
M <- rbind(A, B)
CADM.global(M, 2, 5)
```

`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)
## Fig. 1 in Gittleman (1986):
plot(carnivora$BW ~ carnivora$FW, pch = (1:8)[carnivora$Family], log = "xy",
     xlab = "Female body weight (kg)", ylab = "Birth weigth (g)",
     ylim = c(1, 2000))
legend("bottomright", legend = levels(carnivora$Family), pch = 1:8)
plot(carnivora$BW ~ carnivora$FB, pch = (1:8)[carnivora$Family], log = "xy",
     xlab = "Female brain weight (g)", ylab = "Birth weigth (g)",
     ylim = c(1, 2000))
legend("bottomright", legend = levels(carnivora$Family), pch = 1:8)
```

---

checkAlignment *Check DNA Alignments*

Description

This function performs a series of diagnostics on a DNA alignement.

Usage

```
checkAlignment(x, check.gaps = TRUE, plot = TRUE, what = 1:4)
```

Arguments

x	an object of class "DNAbin".
check.gaps	a logical value specifying whether to check the distribution of alignment gaps.
plot	a logical value specifying whether to do the plots.
what	an integer value giving the plot to be done. By default, four plots are done on the same figure.

Details

This function prints on the console a series of diagnostics on the set a aligned DNA sequences. If alignment gaps are present, their width distribution is analysed, as well as the width of contiguous base segments. The pattern of nucleotide diversity on each site is also analysed, and a relevant table is printed.

If plot = TRUE, four plots are done: an image of the alignement, the distribution of gap widths (if present), the Shannon index of nucleotide diversity along the sequence, and the number of observed bases along the sequence.

If the sequences contain many gaps, it might be better to set check.gaps = FALSE to skip the analysis of contiguous segments.

Value

NULL

Author(s)

Emmanuel Paradis

See Also

alview, image.DNAbin, all.equal.DNAbin

Examples

```
data(woodmouse)
checkAlignment(woodmouse)
layout(1)
```

---

checkLabel *Checking Labels*

Description

Checking and correcting character strings, particularly before writing a Newick tree.

Usage

```
checkLabel(x)
```

Arguments

x	a vector of mode character.

Details

This function deletes the leading and trailing spaces (including tabulations, new lines, and left or right parentheses at the beginning or end of the strings), substitutes the spaces inside the strings by underscores, and substitutes commas, colons, semicolons, and parentheses inside the strings by dashes.

Value

a vector of mode character.

Author(s)

Emmanuel Paradis

See Also

makeLabel, makeNodeLabel, mixedFontLabel, stripLabel, updateLabel

Examples

```
checkLabel(" Homo sapiens\t(Primates; World)   ")
```

checkValidPhylo *Check the Structure of a "phylo" Object*

Description

This function takes as single argument an object (phy), checks its elements, and prints a diagnostic. All problems are printed with a label: FATAL (will likely cause an error or a crash) or MODERATE (may cause some problems).

This function is mainly intended for developers creating "phylo" objects from scratch.

Usage

```
checkValidPhylo(phy)
```

Arguments

phy	an object of class "phylo".

Value

NULL.

Author(s)

Emmanuel Paradis

Examples

```
tr <- rtree(3)
checkValidPhylo(tr)
tr$edge[1] <- 0
checkValidPhylo(tr)
```

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

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(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

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

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")
layout(1)
```

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).

If two nodes are linked (i.e., one is the ancestor of the other) and have the same values of age.min and age.max (say, 10 and 15) this will result in an error because the medians of these values are used as initial times (here 12.5) giving initial branch length(s) equal to zero. The easiest way to solve this is to change slightly the given values, for instance use age.max = 14.9 for the youngest node, or age.max = 15.1 for the oldest one (or similarly for age.min).

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).

Note

The new function chronos replaces the present one which is no more maintained.

Author(s)

Emmanuel Paradis

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

chronos, chronoMPL

---

chronos *Molecular Dating by Penalised Likelihood and Maximum Likelihood*

Description

chronos is the main function fitting a chronogram to a phylogenetic tree whose branch lengths are in number of substitution per sites.

makeChronosCalib is a tool to prepare data frames with the calibration points of the phylogenetic tree.

chronos.control creates a list of parameters to be passed to chronos.

Usage

```
chronos(phy, lambda = 1, model = "correlated", quiet = FALSE,
        calibration = makeChronosCalib(phy),
        control = chronos.control())
## S3 method for class 'chronos'
print(x, ...)
makeChronosCalib(phy, node = "root", age.min = 1,
   age.max = age.min, interactive = FALSE, soft.bounds = FALSE)
chronos.control(...)
```

Arguments

phy	an object of class "phylo".
lambda	value of the smoothing parameter.
model	a character string specifying the model of substitution rate variation among branches. The possible choices are: "correlated", "relaxed", "discrete", "clock", or an unambiguous abbreviation of these.
quiet	a logical value; by default the calculation progress are displayed.
calibration	a data frame (see details).
control	a list of parameters controlling the optimisation procedure (see details).
x	an object of class c("chronos", "phylo").
node	a vector of integers giving the node numbers for which a calibration point is given. The default is a short-cut for the root.

age.min, age.max
: vectors of numerical values giving the minimum and maximum ages of the nodes specified in node.

interactive
: a logical value. If TRUE, then phy is plotted and the user is asked to click close to a node and enter the ages on the keyboard.

soft.bounds
: (currently unused)

...
: in the case of chronos.control: one of the five parameters controlling optimisation (unused in the case of print.chronos).

Details

chronos replaces chronopl but with a different interface and some extensions (see References).

The known dates (argument calibration) must be given in a data frame with the following column names: node, age.min, age.max, and soft.bounds (the last one is yet unused). For each row, these are, respectively: the number of the node in the "phylo" coding standard, the minimum age for this node, the maximum age, and a logical value specifying whether the bounds are soft. If age.min = age.max, this means that the age is exactly known. This data frame can be built with makeChronosCalib which returns by default a data frame with a single row giving age = 1 for the root. The data frame can be built interactively by clicking on the plotted tree.

The argument control allows one to change some parameters of the optimisation procedure. This must be a list with names. The available options with their default values are:

- tol = 1e-8: tolerance for the estimation of the substitution rates.
- iter.max = 1e4: the maximum number of iterations at each optimization step.
- eval.max = 1e4: the maximum number of function evaluations at each optimization step.
- nb.rate.cat = 10: the number of rate categories if model = "discrete" (set this parameter to 1 to fit a strict clock model).
- dual.iter.max = 20: the maximum number of alternative iterations between rates and dates.
- epsilon = 1e-6: the convergence diagnostic criterion.

Using model = "clock" is actually a short-cut to model = "discrete" and setting nb.rate.cat = 1 in the list passed to control.

The command chronos.control() returns a list with the default values of these parameters. They may be modified by passing them to this function, or directly in the list.

Value

chronos returns an object of class c("chronos", "phylo"). There is a print method for it. There are additional attributes which can be visualised with str or extracted with attr.

makeChronosCalib returns a data frame.

chronos.control returns a list.

Author(s)

Emmanuel Paradis, Santiago Claramunt, Guillaume Louvel

References

Kim, J. and Sanderson, M. J. (2008) Penalized likelihood phylogenetic inference: bridging the parsimony-likelihood gap. *Systematic Biology*, **57**, 665–674.

Paradis, E. (2013) Molecular dating of phylogenies by likelihood methods: a comparison of models and a new information criterion. *Molecular Phylogenetics and Evolution*, **67**, 436–444.

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

chronoMPL

Examples

```
library(ape)
tr <- rtree(10)
### the default is the correlated rate model:
chr <- chronos(tr)
### strict clock model:
ctrl <- chronos.control(nb.rate.cat = 1)
chr.clock <- chronos(tr, model = "discrete", control = ctrl)
### How different are the rates?
attr(chr, "rates")
attr(chr.clock, "rates")
## Not run:
cal <- makeChronosCalib(tr, interactive = TRUE)
cal
### if you made mistakes, you can edit the data frame with:
### fix(cal)
chr <- chronos(tr, calibration = cal)

## End(Not run)
```

`clustal` *Multiple Sequence Alignment with External Applications*

Description

These functions call their respective program from R to align a set of nucleotide sequences of class `"DNAbin"` or `"AAbin"`. The application(s) must be installed seperately and it is highly recommended to do this so that the executables are in a directory located on the PATH of the system.

This version includes an experimental version of `muscle5` which calls MUSCLE5 (see the link to the documentation in the References below); `muscle` still calls MUSCLE version 3. Note that the executable of MUSCLE5 is also named 'muscle' by the default compilation setting.

The functions `efastats` and `letterconf` require MUSCLE5.

Usage

```
clustal(x, y, guide.tree, pw.gapopen = 10, pw.gapext = 0.1,
        gapopen = 10, gapext = 0.2, exec = NULL, MoreArgs = "",
        quiet = TRUE, original.ordering = TRUE, file)
clustalomega(x, y, guide.tree, exec = NULL,MoreArgs = "",
             quiet = TRUE, original.ordering = TRUE, file)
muscle(x, y, guide.tree, exec, MoreArgs = "",
        quiet = TRUE, original.ordering = TRUE, file)
muscle5(x, exec = "muscle", MoreArgs = "", quiet = FALSE,
        file, super5 = FALSE, mc.cores = 1)
tcoffee(x, exec = "t_coffee", MoreArgs = "", quiet = TRUE,
        original.ordering = TRUE)

efastats(X, exec = "muscle", quiet = FALSE)
letterconf(X, exec = "muscle")
```

Arguments

x	an object of class "DNAbin" or "AAbin" (can be missing).
y	an object of class "DNAbin" or "AAbin" used for profile alignment (can be missing).
guide.tree	guide tree, an object of class "phylo" (can be missing).
pw.gapopen, pw.gapext	gap opening and gap extension penalties used by Clustal during pairwise alignments.
gapopen, gapext	idem for global alignment.
exec	a character string giving the name of the program, with its path if necessary. clustal tries to guess this argument depending on the operating system (see details).
MoreArgs	a character string giving additional options.
quiet	a logical: the default is to not print on R's console the messages from the external program.
original.ordering	a logical specifying whether to return the aligned sequences in the same order than in x (TRUE by default).
file	a file with its path if results should be stored (can be missing).
super5	a logical value. By default, the PPP algorithm is used.
mc.cores	the number of cores to be used by MUSCLE5.
X	a list with several alignments of the same sequences with all with the same row order.

Details

It is highly recommended to install the executables properly so that they are in a directory located on the PATH (i.e., accessible from any other directory). Alternatively, the full path to the executable

may be given (e.g., `exec = "~/muscle/muscle"`), or a (symbolic) link may be copied in the working directory. For Debian and its derivatives (e.g., Ubuntu), it is recommended to use the binaries distributed by Debian.

`clustal` tries to guess the name of the executable program depending on the operating system. Specifically, the followings are used: "clustalw" under Linux, "clustalw2" under MacOS, and "clustalw2.exe" under Windows. For `clustalomega`, "clustalo[.exe]" is the default on all systems (with no specific path).

When called without arguments (i.e., `clustal()`, ...), the function prints the options of the program which may be passed to `MoreArgs`.

Since **ape** 5.1, `clustal`, `clustalomega`, and `muscle` can align AA sequences as well as DNA sequences.

Value

an object of class `"DNAbin"` or `"AAbin"` with the aligned sequences.

`efastats` returns a data frame.

`letterconf` opens the default Web brower.

Author(s)

Emmanuel Paradis, Franz Krah

References

Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T. J., Higgins, D. G. and Thompson, J. D. (2003) Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Research* **31**, 3497–3500. http://www.clustal.org/

Edgar, R. C. (2004) MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, **32**, 1792–1797. http://www.drive5.com/muscle/muscle_userguide3.8.html

Notredame, C., Higgins, D. and Heringa, J. (2000) T-Coffee: A novel method for multiple sequence alignments. *Journal of Molecular Biology*, **302**, 205–217. https://tcoffee.org/

Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., S\"oding, J., Thompson, J. D. and Higgins, D. G. (2011) Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, **7**, 539. http://www.clustal.org/

https://drive5.com/muscle5/

See Also

`image.DNAbin`, `del.gaps`, `all.equal.DNAbin`, `alex`, `alview`, `checkAlignment`

Examples

```
## Not run:
### display the options:
clustal()
clustalomega()
muscle()
tcoffee()

data(woodmouse)
### open gaps more easily:
clustal(woodmouse, pw.gapopen = 1, pw.gapext = 1)
### T-Coffee requires negative values (quite slow; muscle() is much faster):
tcoffee(woodmouse,  MoreArgs = "-gapopen=-10 -gapext=-2")

## End(Not run)
```

---

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

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

collapse.singles deletes the single nodes (i.e., with a single descendant) in a tree.

has.singles tests for the presence of single node(s) in a tree.

Usage

```
collapse.singles(tree, root.edge = FALSE)
has.singles(tree)
```

Arguments

tree	an object of class "phylo".
root.edge	whether to get the singleton edges from the root until the first bifurcating node and put them as root.edge of the returned tree. By default, this is ignored or if the tree has no edge lengths (see examples).

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis, Klaus Schliep

See Also

plot.phylo, read.tree

Examples

```
## a tree with 3 tips and 3 nodes:
e <- c(4L, 6L, 6L, 5L, 5L, 6L, 1L, 5L, 3L, 2L)
dim(e) <- c(5, 2)
tr <- structure(list(edge = e, tip.label = LETTERS[1:3], Nnode = 3L),
                class = "phylo")
tr
has.singles(tr)
## the following shows that node #4 (ie, the root) is a singleton
## and node #6 is the first bifurcating node
tr$edge
## A bifurcating tree has less nodes than it has tips:
## the following used to fail with ape 4.1 or lower:
plot(tr)
collapse.singles(tr) # only 2 nodes
## give branch lengths to use the 'root.edge' option:
tr$edge.length <- runif(5)
str(collapse.singles(tr, TRUE)) # has a 'root.edge'
```

---

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

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 <dutheil@evolbio.mpg.de>

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:
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)
### 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.

predict returns the predicted (fitted) values of the model.

Usage

```
compar.gee(formula, data = NULL, family = "gaussian", phy, corStruct,
          scale.fix = FALSE, scale.value = 1)
## S3 method for class 'compar.gee'
drop1(object, scope, quiet = FALSE, ...)
## S3 method for class 'compar.gee'
predict(object, newdata = NULL, type = c("link", "response"), ...)
```

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 function 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" (ignored if corStruct is used).
corStruct	a (phylogenetic) correlation structure.
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".
newdata	a data frame with column names matching the variables in the formula of the fitted object (see predict for details).
type	a character string specifying the type of predicted values. By default, the linear (link) prediction is returned.
...	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.

effect.assign	a vector of integers assigning the coefficients to the effects (used by drop1).
nobs	the number of observations.
QIC	the quasilikelihood information criterion as defined by Pan (2001).
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".

predict returns a vector or a data frame if newdata is used.

Note

The calculation of the phylogenetic degrees of freedom is likely to be approximative for non-Brownian correlation structures (this will be refined soon).

The calculation of the quasilikelihood information criterion (QIC) needs to be tested.

Author(s)

Emmanuel Paradis

References

Pan, W. (2001) Akaike's information criterion in generalized estimating equations. *Biometrics*, **57**, 120–125.

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)...)
tr <- "((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);"
tree.primates <- read.tree(text = tr)
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)
```

`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 <julien.claude@umontpellier.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)
x <- "((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);"
tree.primates <- read.tree(text = x)
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 trait optimum) is assumed to change. The node(s) can be specified with their labels if phy has node labels. 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

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

```
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(x <- rnorm(23), bird.orders)
### Much better with a fixed alpha:
compar.ou(x, 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 = 25, 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
##     which follows a chi^2 with df = 1.
```

---

comparePhylo *Compare Two "phylo" Objects*

Description

This function compares two phylogenetic trees, rooted or unrooted, and returns a detailed report of this comparison.

Usage

```
comparePhylo(x, y, plot = FALSE, force.rooted = FALSE,
             use.edge.length = FALSE, commons = TRUE,
             location = "bottomleft", ...)
## S3 method for class 'comparePhylo'
print(x, ...)
```

Arguments

x, y	two objects of class "phylo".
plot	a logical value. If TRUE, the two trees are plotted on the same device and their similarities are shown.
force.rooted	a logical value. If TRUE, the trees are considered rooted even if is.rooted returns FALSE.
use.edge.length	a logical value passed to plot.phylo (see below).

commons	whether to show the splits (the default), or the splits specific to each tree (applies only for unrooted trees).
location	location of where to position the legend.
...	further parameters used by plot.phylo, in function print.comparePhylo unused.

Details

In all cases, the numbers of tips and of nodes and the tip labels are compared.

If both trees are rooted, or if force.rooted = TRUE, the clade compositions of each tree are compared. If both trees are also ultrametric, their branching times are compared.

If both trees are unrooted and have the same number of nodes, the bipartitions (aka splits) are compared.

If plot = TRUE, the edge lengths are not used by default because in some situations with unrooted trees, some splits might not be visible if the corresponding internal edge length is very short. To use edge lengths, set use.edge.length = TRUE.

Value

an object of class "comparePhylo" which is a list with messages from the comparison and, optionally, tables comparing branching times.

Author(s)

Emmanuel Paradis, Klaus Schliep

See Also

all.equal.phylo

Examples

```
## two unrooted trees but force comparison as rooted:
a <- read.tree(text = "(a,b,(c,d));")
b <- read.tree(text = "(a,c,(b,d));")
comparePhylo(a, b, plot = TRUE, force.rooted = TRUE)
## two random unrooted trees:
c <- rtree(5, rooted = FALSE)
d <- rtree(5, rooted = FALSE)
comparePhylo(c, d, plot = TRUE)
```

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 <dutheil@evolbio.mpg.de> and Emmanuel Paradis

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)
```

---

compute.brtime *Compute and Set Branching Times*

Description

This function computes the branch lengths of a tree giving its branching times (aka node ages or heights).

Usage

```
compute.brtime(phy, method = "coalescent", force.positive = NULL)
```

Arguments

phy	an object of class "phylo".
method	either "coalescent" (the default), or a numeric vector giving the branching times.
force.positive	a logical value (see details).

Details

By default, a set of random branching times is generated from a simple coalescent, and the option force.positive is set to TRUE so that no branch length is negative.

If a numeric vector is passed to method, it is taken as the branching times of the nodes with respect to their numbers (i.e., the first element of method is the branching time of the node numbered $n + 1$ [= the root], the second element of the node numbered $n + 2$, and so on), so force.positive is set to FALSE. This may result in negative branch lengths. To avoid this, one should use force.positive = TRUE in which case the branching times are eventually reordered.

Value

An object of class "phylo" with branch lengths and ultrametric.

Author(s)

Emmanuel Paradis

See Also

compute.brlen, branching.times

Examples

```
tr <- rtree(10)
layout(matrix(1:4, 2))
plot(compute.brtime(tr)); axisPhylo()
plot(compute.brtime(tr, force.positive = FALSE)); axisPhylo()
plot(compute.brtime(tr, 1:9)); axisPhylo() # a bit nonsense
plot(compute.brtime(tr, 1:9, TRUE)); axisPhylo()
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 = TRUE, rooted = 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).
rooted	a logical specifying whether the trees should be treated as rooted or not.

Details

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

Until **ape** 5.6-2, the trees passed to this function were implicitly treated as rooted, even when the option rooted = FALSE was used. This is now fixed (see PR65 on GitHub) so that, by default, the trees are explicitly treated as unrooted (even if is.rooted returns TRUE). Thus, it could be that results now differ from previous analyses (setting rooted = TRUE might help to replicate previous results).

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis

References

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

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, fail.if.no.length = FALSE)
```

Arguments

x	an object of class "phylo".
fail.if.no.length	a logical values. If the tree has no branch lengths, these are all fixed to one (with a warning) so the computation is done. If you prefer to catch the case of no branch lengths with an error, set this option to TRUE.

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

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 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 = par("fg"), lwd = par("lwd"), lty = par("lty"),
       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 vector indicating the color to be used for the links; recycled as necessary.
lwd	id. for the width.
lty	id. for the line type.
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`, `rotateConstr`

Examples

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

#creation of the association matrix:
association <- cbind(tree2$tip.label, tree2$tip.label)

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	a one sided formula of the form ~ t, or ~ t | g, specifying the taxa covariate t and, optionally, a grouping factor g. A covariate for this correlation structure must be character valued, with entries matching the tip labels in the phylogenetic tree. When a grouping factor is present in form, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.
fixed	a logical specifying whether gls should estimate γ (the default) or keep it fixed.
object	an (initialized) object of class "corBlomberg".
covariate	an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to getCovariate(object).
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). The exact value has no effect on model fitting with PGLS.
`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`	a one sided formula of the form ~ t, or ~ t | g, specifying the taxa covariate t and, optionally, a grouping factor g. A covariate for this correlation structure must be character valued, with entries matching the tip labels in the phylogenetic tree. When a grouping factor is present in form, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.
`covariate`	an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to getCovariate(object).
`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 the 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 `<dutheil@evolbio.mpg.de>`

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 <dutheil@evolbio.mpg.de>, Emmanuel Paradis

See Also

corClasses and gls in the **nlme** librarie, corBrownian, corMartins, corGrafen, corPagel, corBlomberg, vcv, vcv2phylo

Examples

```
library(nlme)
txt <- "((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);"
tree.primates <- read.tree(text = txt)
X <- c(4.09434, 3.61092, 2.37024, 2.02815, -1.46968)
Y <- c(4.74493, 3.33220, 3.36730, 2.89037, 2.30259)
Species <- c("Homo", "Pongo", "Macaca", "Ateles", "Galago")
dat <- data.frame(Species = Species, X = X, Y = Y)

m1 <- gls(Y ~ X, dat, correlation=corBrownian(1, tree.primates, form = ~Species))
summary(m1)
m2 <- gls(Y ~ X, dat, correlation=corMartins(1, tree.primates, form = ~Species))
summary(m2)
corMatrix(m2$modelStruct$corStruct)
m3 <- gls(Y ~ X, dat, correlation=corGrafen(1, tree.primates, form = ~Species))
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 ρ 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	a one sided formula of the form ~ t, or ~ t | g, specifying the taxa covariate t and, optionally, a grouping factor g. A covariate for this correlation structure must be character valued, with entries matching the tip labels in the phylogenetic tree. When a grouping factor is present in form, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.
covariate	an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to getCovariate(object).
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 <dutheil@evolbio.mpg.de>

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`	a one sided formula of the form ~ t, or ~ t | g, specifying the taxa covariate t and, optionally, a grouping factor g. A covariate for this correlation structure must be character valued, with entries matching the tip labels in the phylogenetic tree. When a grouping factor is present in form, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.

covariate	an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to getCovariate(object).
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 <dutheil@evolbio.mpg.de>

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 λ. 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 λ.
phy	an object of class "phylo".
form	a one sided formula of the form ~ t, or ~ t | g, specifying the taxa covariate t and, optionally, a grouping factor g. A covariate for this correlation structure must be character valued, with entries matching the tip labels in the phylogenetic tree. When a grouping factor is present in form, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.
fixed	a logical specifying whether gls should estimate λ (the default) or keep it fixed.
object	an (initialized) object of class "corPagel".
covariate	an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to getCovariate(object).
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.

corphylo *Correlations among Multiple Traits with Phylogenetic Signal*

Description

This function calculates Pearson correlation coefficients for multiple continuous traits that may have phylogenetic signal, allowing users to specify measurement error as the standard error of trait values at the tips of the phylogenetic tree. Phylogenetic signal for each trait is estimated from the data assuming that trait evolution is given by a Ornstein-Uhlenbeck process. Thus, the function allows the estimation of phylogenetic signal in multiple traits while incorporating correlations among traits. It is also possible to include independent variables (covariates) for each trait to remove possible confounding effects. corphylo() returns the correlation matrix for trait values, estimates of phylogenetic signal for each trait, and regression coefficients for independent variables affecting each trait.

Usage

```
corphylo(X, U = list(), SeM = NULL, phy = NULL, REML = TRUE,
method = c("Nelder-Mead", "SANN"), constrain.d = FALSE, reltol = 10^-6,
maxit.NM = 1000, maxit.SA = 1000, temp.SA = 1, tmax.SA = 1, verbose = FALSE)

## S3 method for class 'corphylo'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

X	a n x p matrix with p columns containing the values for the n taxa. Rows of X should have rownames matching the taxon names in phy.
U	a list of p matrices corresponding to the p columns of X, with each matrix containing independent variables for the corresponding column of X. The rownames of each matrix within U must be the same as X, or alternatively, the order of values in rows must match those in X. If U is omitted, only the mean (aka intercept) for each column of X is estimated. If U[[i]] is NULL, only an intercept is estimated for X[, i]. If all values of U[[i]][j] are the same, this variable is automatically dropped from the analysis (i.e., there is no offset in the regression component of the model).
SeM	a n x p matrix with p columns containing standard errors of the trait values in X. The rownames of SeM must be the same as X, or alternatively, the order of values in rows must match those in X. If SeM is omitted, the trait values are assumed to be known without error. If only some traits have mesurement errors, the remaining traits can be given zero-valued standard errors.
phy	a phylo object giving the phylogenetic tree. The rownames of phy must be the same as X, or alternatively, the order of values in rows must match those in X.
REML	whether REML or ML is used for model fitting.
method	in optim(), either Nelder-Mead simplex minimization or SANN (simulated annealing) minimization is used. If SANN is used, it is followed by Nelder-Mead minimization.

constrain.d	if constrain.d is TRUE, the estimates of d are constrained to be between zero and 1. This can make estimation more stable and can be tried if convergence is problematic. This does not necessarily lead to loss of generality of the results, because before using corphylo, branch lengths of phy can be transformed so that the "starter" tree has strong phylogenetic signal.
reltol	a control parameter dictating the relative tolerance for convergence in the optimization; see optim().
maxit.NM	a control parameter dictating the maximum number of iterations in the optimization with Nelder-Mead minimization; see optim().
maxit.SA	a control parameter dictating the maximum number of iterations in the optimization with SANN minimization; see optim().
temp.SA	a control parameter dictating the starting temperature in the optimization with SANN minimization; see optim().
tmax.SA	a control parameter dictating the number of function evaluations at each temperature in the optimization with SANN minimization; see optim().
verbose	if TRUE, the model logLik and running estimates of the correlation coefficients and values of d are printed each iteration during optimization.
x	an objects of class corphylo.
digits	the number of digits to be printed.
...	arguments passed to and from other methods.

Details

For the case of two variables, the function estimates parameters for the model of the form, for example,

$$X[1] = B[1,0] + B[1,1] * u[1,1] + \epsilon[1]$$
$$X[2] = B[2,0] + B[2,1] * u[2,1] + \epsilon[2]$$
$$\epsilon\, Gaussian(0, V)$$

where $B[1,0]$, $B[1,1]$, $B[2,0]$, and $B[2,1]$ are regression coefficients, and V is a variance-covariance matrix containing the correlation coefficient r, parameters of the OU process $d1$ and $d2$, and diagonal matrices $M1$ and $M2$ of measurement standard errors for $X[1]$ and $X[2]$. The matrix V is $2nx2n$, with nxn blocks given by

$$V[1,1] = C[1,1](d1) + M1$$
$$V[1,2] = C[1,2](d1, d2)$$
$$V[2,1] = C[2,1](d1, d2)$$
$$V[2,2] = C[2,2](d2) + M2$$

where $C[i,j](d1, d2)$ are derived from phy under the assumption of joint OU evolutionary processes for each trait (see Zheng et al. 2009). This formulation extends in the obvious way to more than two traits.

Value

An object of class "corphylo".

cor.matrix	the p x p matrix of correlation coefficients.
d	values of d from the OU process for each trait.
B	estimates of the regression coefficients, including intercepts. Coefficients are named according to the list U. For example, B1.2 is the coefficient corresponding to U[[1]][, 2], and if column 2 in U[[1]] is named "colname2", then the coefficient will be B1.colname2. Intercepts have the form B1.0.
B.se	standard errors of the regression coefficients.
B.cov	covariance matrix for regression coefficients.
B.zscore	Z scores for the regression coefficients.
B.pvalue	tests for the regression coefficients being different from zero.
logLik	he log likelihood for either the restricted likelihood (REML = TRUE) or the overall likelihood (REML = FALSE).
AIC	AIC for either the restricted likelihood (REML = TRUE) or the overall likelihood (REML = FALSE).
BIC	BIC for either the restricted likelihood (REML = TRUE) or the overall likelihood (REML = FALSE).
REML	whether REML is used rather than ML (TRUE or FALSE).
constrain.d	whether or not values of d were constrained to be between 0 and 1 (TRUE or FALSE).
XX	values of X in vectorized form, with each trait X[, i] standardized to have mean zero and standard deviation one.
UU	design matrix with values in UU corresponding to XX; each variable U[[i]][, j] is standardized to have mean zero and standard deviation one.
MM	vector of measurement standard errors corresponding to XX, with the standard errors suitably standardized.
Vphy	the phylogenetic covariance matrix computed from phy and standardized to have determinant equal to one.
R	covariance matrix of trait values relative to the standardized values of XX.
V	overall estimated covariance matrix of residuals for XX including trait correlations, phylogenetic signal, and measurement error variances. This matrix can be used to simulate data for parametric bootstrapping. See examples.
C	matrix V excluding measurement error variances.
convcode	he convergence code provided by optim().
niter	number of iterations performed by optim().

Author(s)

Anthony R. Ives

References

Zheng, L., A. R. Ives, T. Garland, B. R. Larget, Y. Yu, and K. F. Cao. 2009. New multivariate tests for phylogenetic signal and trait correlations applied to ecophysiological phenotypes of nine *Manglietia* species. *Functional Ecology* **23**:1059–1069.

Examples

```
## Simple example using data without correlations or phylogenetic
## signal. This illustrates the structure of the input data.

phy <- rcoal(10, tip.label = 1:10)
X <- matrix(rnorm(20), nrow = 10, ncol = 2)
rownames(X) <- phy$tip.label
U <- list(NULL, matrix(rnorm(10, mean = 10, sd = 4), nrow = 10, ncol = 1))
rownames(U[[2]]) <- phy$tip.label
SeM <- matrix(c(0.2, 0.4), nrow = 10, ncol = 2)
rownames(SeM) <- phy$tip.label

corphylo(X = X, SeM = SeM, U = U, phy = phy, method = "Nelder-Mead")

## Not run:
## Simulation example for the correlation between two variables. The
## example compares the estimates of the correlation coefficients from
## corphylo when measurement error is incorporated into the analyses with
## three other cases: (i) when measurement error is excluded, (ii) when
## phylogenetic signal is ignored (assuming a "star" phylogeny), and (iii)
## neither measurement error nor phylogenetic signal are included.

## In the simulations, variable 2 is associated with a single
## independent variable. This requires setting up a list U that has 2
## elements: element U[[1]] is NULL and element U[[2]] is a n x 1 vector
## containing simulated values of the independent variable.

# Set up parameter values for simulating data
n <- 50
phy <- rcoal(n, tip.label = 1:n)

R <- matrix(c(1, 0.7, 0.7, 1), nrow = 2, ncol = 2)
d <- c(0.3, .95)
B2 <- 1

Se <- c(0.2, 1)
SeM <- matrix(Se, nrow = n, ncol = 2, byrow = T)
rownames(SeM) <- phy$tip.label

# Set up needed matrices for the simulations
p <- length(d)

star <- stree(n)
star$edge.length <- array(1, dim = c(n, 1))
star$tip.label <- phy$tip.label
```

```
Vphy <- vcv(phy)
Vphy <- Vphy/max(Vphy)
Vphy <- Vphy/exp(determinant(Vphy)$modulus[1]/n)

tau <- matrix(1, nrow = n, ncol = 1)
C <- matrix(0, nrow = p * n, ncol = p * n)
for (i in 1:p) for (j in 1:p) {
 Cd <- (d[i]^tau * (d[j]^t(tau)) * (1 - (d[i] * d[j])^Vphy))/(1 - d[i] * d[j])
 C[(n * (i - 1) + 1):(i * n), (n * (j - 1) + 1):(j * n)] <- R[i, j] * Cd
}
MM <- matrix(SeM^2, ncol = 1)
V <- C + diag(as.numeric(MM))

## Perform a Cholesky decomposition of Vphy. This is used to generate
## phylogenetic signal: a vector of independent normal random variables,
## when multiplied by the transpose of the Cholesky deposition of Vphy will
## have covariance matrix equal to Vphy.
iD <- t(chol(V))

# Perform Nrep simulations and collect the results
Nrep <- 100
cor.list <- matrix(0, nrow = Nrep, ncol = 1)
cor.noM.list <- matrix(0, nrow = Nrep, ncol = 1)
cor.noP.list <- matrix(0, nrow = Nrep, ncol = 1)
cor.noMP.list <- matrix(0, nrow = Nrep, ncol = 1)
d.list <- matrix(0, nrow = Nrep, ncol = 2)
d.noM.list <- matrix(0, nrow = Nrep, ncol = 2)
B.list <- matrix(0, nrow = Nrep, ncol = 3)
B.noM.list <- matrix(0, nrow = Nrep, ncol = 3)
B.noP.list <- matrix(0, nrow = Nrep, ncol = 3)
for (rep in 1:Nrep) {
 XX <- iD
 X <- matrix(XX, nrow = n, ncol = 2)
 rownames(X) <- phy$tip.label

 U <- list(NULL, matrix(rnorm(n, mean = 2, sd = 10), nrow = n, ncol = 1))
 rownames(U[[2]]) <- phy$tip.label
 colnames(U[[2]]) <- "V1"
 X[,2] <- X[,2] + B2[1] * U[[2]][,1] - B2[1] * mean(U[[2]][,1])

 z <- corphylo(X = X, SeM = SeM, U = U, phy = phy, method = "Nelder-Mead")
 z.noM <- corphylo(X = X, U = U, phy = phy, method = "Nelder-Mead")
 z.noP <- corphylo(X = X, SeM = SeM, U = U, phy = star, method = "Nelder-Mead")

 cor.list[rep] <- z$cor.matrix[1, 2]
 cor.noM.list[rep] <- z.noM$cor.matrix[1, 2]
 cor.noP.list[rep] <- z.noP$cor.matrix[1, 2]
 cor.noMP.list[rep] <- cor(cbind(lm(X[,1] ~ 1)$residuals, lm(X[,2] ~ U[[2]])$residuals))[1,2]

 d.list[rep, ] <- z$d
 d.noM.list[rep, ] <- z.noM$d

 B.list[rep, ] <- z$B
```

```
 B.noM.list[rep, ] <- z.noM$B
 B.noP.list[rep, ] <- z.noP$B

 show(c(rep, z$convcode, z$cor.matrix[1, 2], z$d))
}
correlation <- rbind(R[1, 2], mean(cor.list), mean(cor.noM.list),
                     mean(cor.noP.list), mean(cor.noMP.list))
rownames(correlation) <- c("True", "With SeM and Phy", "Without SeM",
                           "Without Phy", "Without Phy or SeM")
correlation

signal.d <- rbind(d, colMeans(d.list), colMeans(d.noM.list))
rownames(signal.d) <- c("True", "With SeM and Phy", "Without SeM")
signal.d

est.B <- rbind(c(0, 0, B2), colMeans(B.list), colMeans(B.noM.list),
               colMeans(B.noP.list))
rownames(est.B) <- c("True", "With SeM and Phy", "Without SeM", "Without Phy")
colnames(est.B) <- rownames(z$B)
est.B

# Example simulation output
# correlation
                        # [,1]
# True                0.7000000
# With SeM and Phy    0.7055958
# Without SeM         0.3125253
# Without Phy         0.4054043
# Without Phy or SeM 0.3476589

# signal.d
                      # [,1]      [,2]
# True              0.300000 0.9500000
# With SeM and Phy 0.301513 0.9276663
# Without SeM      0.241319 0.4872675

# est.B
                         # B1.0      B2.0     B2.V1
# True               0.00000000 0.0000000 1.0000000
# With SeM and Phy -0.01285834 0.2807215 0.9963163
# Without SeM        0.01406953 0.3059110 0.9977796
# Without Phy        0.02139281 0.3165731 0.9942140

## End(Not run)
```

`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 <dutheil@evolbio.mpg.de> and Emmanuel Paradis

See Also

plot.correlogram, Moran.I

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.

---

dbd *Probability Density Under Birth–Death Models*

Description

These functions compute the probability density under some birth–death models, that is the probability of obtaining x species after a time t giving how speciation and extinction probabilities vary through time (these may be constant, or even equal to zero for extinction).

Usage

```
dyule(x, lambda = 0.1, t = 1, log = FALSE)
dbd(x, lambda, mu, t, conditional = FALSE, log = FALSE)
dbdTime(x, birth, death, t, conditional = FALSE,
        BIRTH = NULL, DEATH = NULL, fast = FALSE)
```

Arguments

x	a numeric vector of species numbers (see Details).
lambda	a numerical value giving the probability of speciation; can be a vector with several values for dyule.
mu	id. for extinction.
t	id. for the time(s).
log	a logical value specifying whether the probabilities should be returned log-transformed; the default is FALSE.
conditional	a logical specifying whether the probabilities should be computed conditional under the assumption of no extinction after time t.
birth, death	a (vectorized) function specifying how the speciation or extinction probability changes through time (see yule.time and below).
BIRTH, DEATH	a (vectorized) function giving the primitive of birth or death.
fast	a logical value specifying whether to use faster integration (see bd.time).

Details

These three functions compute the probabilities to observe x species starting from a single one after time t (assumed to be continuous). The first function is a short-cut for the second one with mu = 0 and with default values for the two other arguments. dbdTime is for time-varying lambda and mu specified as R functions.

dyule is vectorized simultaneously on its three arguments x, lambda, and t, according to R's rules of recycling arguments. dbd is vectorized simultaneously x and t (to make likelihood calculations easy), and dbdTime is vectorized only on x; the other arguments are eventually shortened with a warning if necessary.

The returned value is, logically, zero for values of x out of range, i.e., negative or zero for dyule or if conditional = TRUE. However, it is not checked if the values of x are positive non-integers and the probabilities are computed and returned.

The details on the form of the arguments birth, death, BIRTH, DEATH, and fast can be found in the links below.

Value

a numeric vector.

Note

If you use these functions to calculate a likelihood function, it is strongly recommended to compute the log-likelihood with, for instance in the case of a Yule process, sum(dyule(, log = TRUE)) (see examples).

Author(s)

Emmanuel Paradis

References

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

See Also

bd.time, yule.time

Examples

```
x <- 0:10
plot(x, dyule(x), type = "h", main = "Density of the Yule process")
text(7, 0.85, expression(list(lambda == 0.1, t == 1)))

y <- dbd(x, 0.1, 0.05, 10)
z <- dbd(x, 0.1, 0.05, 10, conditional = TRUE)
d <- rbind(y, z)
colnames(d) <- x
barplot(d, beside = TRUE, ylab = "Density", xlab = "Number of species",
        legend = c("unconditional", "conditional on\nno extinction"),
        args.legend = list(bty = "n"))
title("Density of the birth-death process")
text(17, 0.4, expression(list(lambda == 0.1, mu == 0.05, t == 10)))

## Not run:
### generate 1000 values from a Yule process with lambda = 0.05
x <- replicate(1e3, Ntip(rlineage(0.05, 0)))

### the correct way to calculate the log-likelihood...:
sum(dyule(x, 0.05, 50, log = TRUE))

### ... and the wrong way:
log(prod(dyule(x, 0.05, 50)))

### a third, less preferred, way:
sum(log(dyule(x, 0.05, 50)))

## End(Not run)
```

---

def *Definition of Vectors for Plotting or Annotating*

Description

This function can be used to define vectors to annotate a set of taxon names, labels, etc. It should facilitate the (re)definition of colours or similar attributes for plotting trees or other graphics.

Usage

```
def(x, ..., default = NULL, regexp = FALSE)
```

Arguments

x	a vector of mode character.
...	a series of statements defining the attributes.
default	the default to be used (see details).
regexp	a logical value specifying whether the statements defined in ... should be taken as regular expressions.

Details

The idea of this function is to make the definition of colours, etc., simpler than what is done usually. A typical use is:

```
def(tr$tip.label, Homo_sapiens = "blue")
```

which will return a vector of character strings all "black" except one matching the tip label "Homo_sapiens" which will be "blue". Another use could be:

```
def(tr$tip.label, Homo_sapiens = 2)
```

which will return a vector a numerical values all 1 except for "Homo_sapiens" which will be 2. Several definitions can be done, e.g.:

```
def(tr$tip.label, Homo_sapiens = "blue", Pan_paniscus = "red")
```

The default value is determined with respect to the mode of the values given with the ... (either "black" or 1).

If `regexp = TRUE` is used, then the names of the statements must be quoted, e.g.:

```
def(tr$tip.label, "^Pan_" = "red", regexp = TRUE)
```

will return "red" for all labels starting with "Pan_".

Value

a vector of the same length than x.

Author(s)

Emmanuel Paradis

Examples

```
data(bird.orders)
a <- def(bird.orders$tip.label, Galliformes = 2)
str(a) # numeric
plot(bird.orders, font = a)
co <- def(bird.orders$tip.label, Passeriformes = "red", Trogoniformes = "blue")
str(co) # character
plot(bird.orders, tip.color = co)
### use of a regexp (so we need to quote it) to colour all orders
### with names starting with "C" (and change the default):
co2 <- def(bird.orders$tip.label, "^C" = "gold", default = "grey", regexp = TRUE)
plot(bird.orders, tip.color = co2)
```

degree *Vertex Degrees in Trees and Networks*

Description

degree is a generic function to calculate the degree of all nodes in a tree or in a network.

Usage

```
degree(x, ...)
## S3 method for class 'phylo'
degree(x, details = FALSE, ...)
## S3 method for class 'evonet'
degree(x, details = FALSE, ...)
```

Arguments

x	an object (tree, network, ...).
details	whether to return the degree of each node in the tree, or a summary table (the default).
...	arguments passed to methods.

Details

The degree of a node (or vertex) in a network is defined by the number of branches (or edges) that connect to this node. In a phylogenetic tree, the tips (or terminal nodes) are of degree one, and the (internal) nodes are of degree two or more.

There are currently two methods for the classes "phylo" and "evonet". The default of these functions is to return a summary table with the degrees observed in the tree or network in the first column, and the number of nodes in the second column. If details = TRUE, a vector giving the degree of each node (as numbered in the edge matrix) is returned.

The validity of the object is not checked, so degree can be used to check problems with badly conformed trees.

Value

a data frame if details = FALSE, or a vector of integers otherwise.

Author(s)

Emmanuel Paradis

See Also

checkValidPhylo

Examples

```
data(bird.orders)
degree(bird.orders)
degree(bird.orders, details = TRUE)

data(bird.families)
degree(bird.families)

degree(rtree(10)) # 10, 1, 8
degree(rtree(10, rooted = FALSE)) # 10, 0, 8
degree(stree(10)) # 10 + 1 node of degree 10
```

---

del.gaps *Delete Alignment Gaps in DNA or AA Sequences*

Description

These functions remove gaps ("-") in a sample of DNA sequences.

Usage

```
del.gaps(x)
del.colgapsonly(x, threshold = 1, freq.only = FALSE)
del.rowgapsonly(x, threshold = 1, freq.only = FALSE)
```

Arguments

x	a matrix, a list, or a vector containing the DNA or AA sequences; only matrices for del.colgapsonly and for del.rowgapsonly.
threshold	the largest gap proportion to delete the column or row.
freq.only	if TRUE, returns only the numbers of gaps for each column or row.

Details

del.gaps remove all gaps, so the returned sequences may not have all the same lengths and are therefore returned in a list.

del.colgapsonly removes the columns with a proportion at least threshold of gaps. Thus by default, only the columns with gaps only are removed (useful when a small matrix is extracted from a large alignment). del.rowgapsonly does the same for the rows.

The class of the input sequences is respected and kept unchanged, unless it contains neither "DNAbin" nor "AAbin" in which case the object is first converted into the class "DNAbin".

Value

del.gaps returns a vector (if there is only one input sequence) or a list of sequences; del.colgapsonly and del.rowgapsonly return a matrix of sequences or a numeric vector (with names for the second function) if freq.only = TRUE.

Author(s)

Emmanuel Paradis

See Also

base.freq, seg.sites, image.DNAbin, checkAlignment

---

delta.plot *Delta Plots*

Description

This function makes a δ plot following Holland et al. (2002).

Usage

```
delta.plot(X, k = 20, plot = TRUE, which = 1:2)
```

Arguments

X	a distance matrix, may be an object of class "dist".
k	an integer giving the number of intervals in the plot.
plot	a logical specifying whether to draw the δ plot (the default).
which	a numeric vector indicating which plots are done; 1: the histogram of the δ_q values, 2: the plot of the individual $\bar{\delta}$ values. By default, both plots are done.

Details

See Holland et al. (2002) for details and interpretation.

The computing time of this function is proportional to the fourth power of the number of observations ($O(n^4)$), so calculations may be very long with only a slight increase in sample size.

Value

This function returns invisibly a named list with two components:

- counts: the counts for the histogram of δ_q values
- delta.bar: the mean δ value for each observation

Author(s)

Emmanuel Paradis

References

Holland, B. R., Huber, K. T., Dress, A. and Moulton, V. (2002) Delta plots: a tool for analyzing phylogenetic distance data. *Molecular Biology and Evolution*, **12**, 2051–2059.

See Also

dist.dna

Examples

```
data(woodmouse)
d <- dist.dna(woodmouse)
delta.plot(d)
layout(1)
delta.plot(d, 40, which = 1)
```

---

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; this must be of class "DNAbin" (use as.DNAbin is they are stored as character).
model	a character string specifying the evolutionary model to be used; must be one of "raw", "N", "TS", "TV", "JC69", "K80" (the default), "F81", "K81", "F84", "BH87", "T92", "TN93", "GG95", "logdet", "paralin", "indel", or "indelblock".
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 possibly used to apply a correction to the distances (by default 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 (ignored if model = "indel" or "indelblock").
base.freq	the base frequencies to be used in the computations (if applicable). By default, the base frequencies are computed from the whole set 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`, `N`: This is simply the proportion or the number 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` may have.
- `TS`, `TV`: These are the numbers of transitions and transversions, respectively.
- `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. Formulae from Gu and Li (1996) are used. `dist.logdet` in **phangorn** uses a different implementation that gives substantially different distances for low-diverging sequences.
- `paralin`: Lake (1994) developed the paralinear distance which can be viewed as another variant of the Barry–Hartigan distance.
- `indel`: this counts the number of sites where there is an insertion/deletion gap in one sequence and not in the other.
- `indelblock`: same than before but contiguous gaps are counted as a single unit. Note that the distance between `-A-` and `A--` is 3 because there are three different blocks of gaps, whereas the "indel" distance will be 2.

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.

Note

If the sequences are very different, most evolutionary distances are undefined and a non-finite value (Inf or NaN) is returned. You may do `dist.dna(, model = "raw")` to check whether some values are higher than 0.75.

Author(s)

Emmanuel Paradis

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.

Gu, X. and Li, W.-H. (1996) Bias-corrected paralinear and LogDet distances and tests of molecular clocks and phylogenies under nonstationary nucleotide frequencies. *Molecular Biology and Evolution*, **13**, 1375–1383.

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

This function computes a matrix of distances between pairs of individuals from a matrix or a data frame of genetic data.

Usage

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

Arguments

`x`	a matrix or a data frame (will be coerced as a matrix).
`method`	a character string specifying the method used to compute the distances; two choices are available: `"pairwise"` and `"percentage"`, or any unambiguous abbreviation of these.
`pairwise.deletion`	a logical indicating whether to delete the columns with missing data on a pairwise basis. The default is to delete the columns with at least one missing observation.

`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, SNP, DNA sequences, ...). 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

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

Note

Missing data (`NA`) are coded and treated in R's usual way.

Author(s)

Emmanuel Paradis

See Also

`dist.dna`, `cophenetic.phylo`, `dist`

---

dist.topo *Topological Distances Between Two Trees*

Description

This function computes the topological distance between two phylogenetic trees or among trees in a list (if `y = NULL` using different methods.

Usage

```
dist.topo(x, y = NULL, method = "PH85", mc.cores = 1)
```

Arguments

x	an object of class "phylo" or of class "multiPhylo".
y	an (optional) object of class "phylo".
method	a character string giving the method to be used: either "PH85", or "score".
mc.cores	the number of cores (CPUs) to be used (passed to **parallel**).

Details

Two methods are available: the one by Penny and Hendy (1985, originally from Robinson and Foulds 1981), and the branch length score by Kuhner and Felsenstein (1994). The trees are always considered as unrooted.

The topological distance is defined as twice the number of internal branches defining different bipartitions of the tips (Robinson and Foulds 1981; Penny and Hendy 1985). Rzhetsky and Nei (1992) proposed a modification of the original formula to take multifurcations into account.

The branch length score may be seen as similar to the previous distance but taking branch lengths into account. Kuhner and Felsenstein (1994) proposed to calculate the square root of the sum of the squared differences of the (internal) branch lengths defining similar bipartitions (or splits) in both trees.

Value

a single numeric value if both x and y are used, an object of class "dist" otherwise.

Note

The geodesic distance of Billera et al. (2001) has been disabled: see the package **distory** on CRAN.

Author(s)

Emmanuel Paradis

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.

Kuhner, M. K. and Felsenstein, J. (1994) Simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, **11**, 459–468.

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.

Robinson, D. F. and Foulds, L. R. (1981) Comparison of phylogenetic trees. *Mathematical Biosciences*, **53**, 131–147.

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

cophenetic.phylo, prop.part

Examples

```
ta <- rtree(30, rooted = FALSE)
tb <- rtree(30, rooted = FALSE)
dist.topo(ta, ta) # 0
dist.topo(ta, tb) # unlikely to be 0

## rmtopology() simulated unrooted trees by default:
TR <- rmtopology(100, 10)
## these trees have 7 internal branches, so the maximum distance
## between two of them is 14:
DTR <- dist.topo(TR)
table(DTR)
```

---

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

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

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

`diversity.contrast.test` *Diversity Contrast Test*

Description

This function performs the diversity contrast test comparing pairs of sister-clades.

Usage

```
diversity.contrast.test(x, method = "ratiolog",
        alternative = "two.sided", nrep = 0, ...)
```

Arguments

`x`	a matrix or a data frame with at least two columns: the first one gives the number of species in clades with a trait supposed to increase or decrease diversification rate, and the second one the number of species in the sister-clades without the trait. Each row represents a pair of sister-clades.
`method`	a character string specifying the kind of test: `"ratiolog"` (default), `"proportion"`, `"difference"`, `"logratio"`, or any unambiguous abbreviation of these.
`alternative`	a character string defining the alternative hypothesis: `"two.sided"` (default), `"less"`, `"greater"`, or any unambiguous abbreviation of these.
`nrep`	the number of replications of the randomization test; by default, a Wilcoxon test is done.
`...`	arguments passed to the function `wilcox.test`.

Details

If `method = "ratiolog"`, the test described in Barraclough et al. (1996) is performed. If `method = "proportion"`, the version in Barraclough et al. (1995) is used. If `method = "difference"`, the signed difference is used (Sargent 2004). If `method = "logratio"`, then this is Wiegmann et al.'s (1993) version. These four tests are essentially different versions of the same test (Vamosi and Vamosi 2005, Vamosi 2007). See Paradis (2012) for a comparison of their statistical performance with other tests.

If `nrep = 0`, a Wilcoxon test is done on the species diversity contrasts with the null hypothesis is that they are distributed around zero. If `nrep > 0`, a randomization procedure is done where the signs of the diversity contrasts are randomly chosen. This is used to create a distribution of the test statistic which is compared with the observed value (the sum of the diversity contrasts).

Value

a single numeric value with the *P*-value.

Author(s)

Emmanuel Paradis

References

Barraclough, T. G., Harvey, P. H. and Nee, S. (1995) Sexual selection and taxonomic diversity in passerine birds. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **259**, 211–215.

Barraclough, T. G., Harvey, P. H., and Nee, S. (1996) Rate of *rbc*L gene sequence evolution and species diversification in flowering plants (angiosperms). *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **263**, 589–591.

Paradis, E. (2012) Shift in diversification in sister-clade comparisons: a more powerful test. *Evolution*, **66**, 288–295.

Sargent, R. D. (2004) Floral symmetry affects speciation rates in angiosperms. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, **271**, 603–608.

Vamosi, S. M. (2007) Endless tests: guidelines for analysing non-nested sister-group comparisons. An addendum. *Evolutionary Ecology Research*, **9**, 717.

Vamosi, S. M. and Vamosi, J. C. (2005) Endless tests: guidelines for analysing non-nested sister-group comparisons. *Evolutionary Ecology Research*, **7**, 567–579.

Wiegmann, B., Mitter, C. and Farrell, B. 1993. Diversification of carnivorous parasitic insects: extraordinary radiation or specialized dead end? *American Naturalist*, **142**, 737–754.

See Also

`slowinskiguyer.test`, `mcconwaysims.test` `richness.yule.test`

Examples

```
### data from Vamosi & Vamosi (2005):
fleshy <- c(1, 1, 1, 1, 1, 3, 3, 5, 9, 16, 33, 40, 50, 100, 216, 393, 850, 947,1700)
dry <- c(2, 64, 300, 89, 67, 4, 34, 10, 150, 35, 2, 60, 81, 1, 3, 1, 11, 1, 18)
x <- cbind(fleshy, dry)
diversity.contrast.test(x)
diversity.contrast.test(x, alt = "g")
diversity.contrast.test(x, alt = "g", nrep = 1e4)
slowinskiguyer.test(x)
mcconwaysims.test(x)
```

`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, printlen = 6, digits = 3, ...)
## S3 method for class 'DNAbin'
rbind(...)
## S3 method for class 'DNAbin'
cbind(..., check.names = TRUE, fill.with.gaps = FALSE,
             quiet = FALSE)
## S3 method for class 'DNAbin'
x[i, j, drop = FALSE]
## S3 method for class 'DNAbin'
as.matrix(x, ...)
## S3 method for class 'DNAbin'
c(..., recursive = FALSE)
## S3 method for class 'DNAbin'
as.list(x, ...)
## S3 method for class 'DNAbin'
labels(object, ...)
```

Arguments

x, object	an object of class "DNAbin".
...	either further arguments to be passed to or from other methods in the case of print, as.matrix, and labels, or a series of objects of class "DNAbin" in the case of rbind, cbind, and c.
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).
fill.with.gaps	a logical indicating whether to keep all possible individuals as indicating by the rownames, and eventually filling the missing data with insertion gaps (ignored if check.names = FALSE).
quiet	a logical to switch off warning messages when some rows are dropped.
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 returned object is of the lowest possible dimension.
recursive	for compatibility with the generic (unused).

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. Note that the default of drop is not the same than the generic operator: this is to avoid dropping rownames when selecting a single sequence.

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`, the default behaviour is to keep only individuals (as indicated by the rownames) for which there are no missing data. If `fill.with.gaps = TRUE`, a 'complete' matrix is returned, eventually with insertion gaps as missing data. If `check.names = TRUE` (the default), the rownames of each matrix are checked, and the rows are reordered if necessary (if some rownames are duplicated, an error is returned). If `check.names = FALSE`, the matrices must all have the same number of rows, and are simply binded; the rownames of the first matrix are used. See the examples.

`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. `as.list` does the reverse operation.

Value

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

Author(s)

Emmanuel Paradis

References

Paradis, E. (2007) A Bit-Level Coding Scheme for Nucleotides. https://emmanuelparadis.github.io/misc/BitLevelCodingScheme_20April2007.pdf

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

See Also

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

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

Examples

```
data(woodmouse)
woodmouse
print(woodmouse, 15, 6)
print(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, ])
### cbind and its options:
x <- woodmouse[1:2, 1:5]
y <- woodmouse[2:4, 6:10]
as.character(cbind(x, y)) # gives warning
as.character(cbind(x, y, fill.with.gaps = TRUE))
## Not run:
as.character(cbind(x, y, check.names = FALSE)) # gives an error

## End(Not run)
```

DNAbin2indel *Recode Blocks of Indels*

Description

This function scans a set of aligned DNA sequences and returns a matrix with information of the localisations and lengths on alignment gaps.

Usage

```
DNAbin2indel(x)
```

Arguments

x	an object of class "DNAbin".

Details

The output matrix has the same dimensions than the input one with, either a numeric value where an alignment gap starts giving the length of the gap, or zero. The rownames are kept.

Value

a numeric matrix.

Author(s)

Emmanuel Paradis

See Also

DNAbin, as.DNAbin, del.gaps, seg.sites, image.DNAbin, checkAlignment

dnds *dN/dS Ratio*

Description

This function computes the pairwise ratios dN/dS for a set of aligned DNA sequences using Li's (1993) method.

Usage

```
dnds(x, code = 1, codonstart = 1, quiet = FALSE,
     details = FALSE, return.categories = FALSE)
```

Arguments

x	an object of class "DNAbin" (matrix or list) with the aligned sequences.
code	an integer value giving the genetic code to be used. Currently, the codes 1 to 6 are supported.
codonstart	an integer giving where to start the translation. This should be 1, 2, or 3, but larger values are accepted and have for effect to start the translation further within the sequence.
quiet	single logical value: whether to indicate progress of calculations.
details	single logical value (see details).
return.categories	a logical value: if TRUE, a matrix of the same size than x is returned giving the degeneracy category of each base in the original alignment.

Details

Since **ape** 5.6, the degeneracy of each codon is calculated directly from the genetic code using the function trans. A consequence is that ambiguous bases are ignored (see solveAmbiguousBases).

If details = TRUE, a table is printed for each pair of sequences giving the numbers of transitions and transversions for each category of degeneracy (nondegenerate, twofold, and fourfold). This is helpful when non-meaningful values are returned (e.g., NaN, Inf, negative values).

Value

an object of class "dist", or a numeric matrix if return.categories = TRUE.

Author(s)

Emmanuel Paradis

References

Li, W.-H. (1993) Unbiased estimation of the rates of synonymous and nonsynonymous substitution. *Journal of Molecular Evolution*, **36**, 96–99.

See Also

AAbin, trans, alview, solveAmbiguousBases

Examples

```
data(woodmouse)
res <- dnds(woodmouse, quiet = TRUE) # NOT correct
res2 <- dnds(woodmouse, code = 2, quiet = TRUE) # using the correct code
identical(res, res2) # FALSE...
cor(res, res2) # ... but very close
## There a few N's in the woodmouse data, but this does not affect
## greatly the results:
res3 <- dnds(solveAmbiguousBases(woodmouse), code = 2, quiet = TRUE)
```

```
cor(res, res3)

## a simple example showing the usefulness of 'details = TRUE'
X <- as.DNAbin(matrix(c("C", "A", "G", "G", "T", "T"), 2, 3))
alview(X)
dnds(X, quiet = TRUE) # NaN
dnds(X, details = TRUE) # only a TV at a nondegenerate site
```

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

Description

`drop.tip` removes the terminal branches of a phylogenetic tree, possibly removing the corresponding internal branches. `keep.tip` does the opposite operation (i.e., returns the induced tree).

`extract.clade` does the inverse operation: it keeps all the tips from a given node, and deletes all the other tips.

Usage

```
drop.tip(phy, tip, ...)
## S3 method for class 'phylo'
drop.tip(phy, tip, trim.internal = TRUE, subtree = FALSE,
         root.edge = 0, rooted = is.rooted(phy), collapse.singles = TRUE,
         interactive = FALSE, ...)
## S3 method for class 'multiPhylo'
drop.tip(phy, tip, ...)

keep.tip(phy, tip, ...)
## S3 method for class 'phylo'
keep.tip(phy, tip, ...)
## S3 method for class 'multiPhylo'
keep.tip(phy, tip, ...)

extract.clade(phy, node, root.edge = 0, collapse.singles = TRUE,
              interactive = FALSE)
```

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`.

rooted	a logical indicating whether the tree must be treated as rooted or not. This allows to force the tree to be considered as unrooted (see examples). See details about a possible root.edge element in the tree.
collapse.singles	a logical specifying whether to delete the internal nodes of degree 2.
node	a node number or label.
interactive	if TRUE the user is asked to select the tips or the node by clicking on the tree which must be plotted.
...	arguments passed from and to methods.

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.

This also applies to node, but if this argument is character and the tree has no node label, this results in an error. If more than one value is given with node (i.e., a vector of length two or more), only the first one is used with a warning.

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 named with node labels if available. Otherwise it is indicated 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. If rooted = FALSE and the tree has a root edge, the latter is removed in the output.

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis, Klaus Schliep, Joseph Brown

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))
plot(drop.tip(bird.orders, c(1:5, 20:23), subtree = TRUE))
plot(drop.tip(bird.orders, c(1:20, 23), subtree = TRUE))
plot(drop.tip(bird.orders, c(1:20, 23), subtree = TRUE, rooted = FALSE))
### 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;
```

---

edges *Draw Additional Edges on a Plotted Tree*

Description

edges draws edges on a plotted tree. fancyarrows enhances arrows with triangle and harpoon heads; it can be called from edges.

Usage

```
edges(nodes0, nodes1, arrows = 0, type = "classical", ...)
fancyarrows(x0, y0, x1, y1, length = 0.25, angle = 30, code = 2,
            col = par("fg"), lty = par("lty"), lwd = par("lwd"),
            type = "triangle", ...)
```

Arguments

nodes0, nodes1	vectors of integers giving the tip and/or node numbers where to start and to end the edges (eventually recycled).
arrows	an integer between 0 and 3; 0: lines (the default); 1: an arrow head is drawn at nodes0; 2: at nodes1; 3: both.
type	if the previous argument is not 0, the type of arrow head: "classical" (just lines, the default), "triangle", "harpoon", or any unambiguous abbreviations of these. For fancyarrows only the last two are available.
x0, y0, x1, y1	the coordinates of the start and end points for fancyarrows (these are not recycled and so should be vectors of the same length).

`length`, `angle`, `code`, `col`, `lty`, `lwd`
: default options similar to those of `arrows`.

`...`
: further arguments passed to `segments`.

Details

The first function is helpful when drawing reticulations on a phylogeny, especially if computed from the edge matrix.

`fancyarrows` does not work with log-transformed scale(s).

Author(s)

Emmanuel Paradis

See Also

`plot.phylo`, `nodelabels`

Examples

```
set.seed(2)
tr <- rcoal(6)
plot(tr, "c")
edges(10, 9, col = "red", lty = 2)
edges(10:11, 8, col = c("blue", "green")) # recycling of 'nodes1'
edges(1, 2, lwd = 2, type = "h", arrows = 3, col = "green")
nodelabels()
```

---

evonet *Evolutionary Networks*

Description

`evonet` builds a network from a tree of class `"phylo"`. There are `print`, `plot`, and `reorder` methods as well as a few conversion functions.

Usage

```
evonet(phy, from, to = NULL)
## S3 method for class 'evonet'
print(x, ...)
## S3 method for class 'evonet'
plot(x, col = "blue", lty = 1, lwd = 1, alpha = 0.5,
              arrows = 0, arrow.type = "classical", ...)
## S3 method for class 'evonet'
Nedge(phy)
## S3 method for class 'evonet'
reorder(x, order = "cladewise", index.only = FALSE, ...)
```

```
## S3 method for class 'evonet'
as.phylo(x, ...)
## S3 method for class 'evonet'
as.networx(x, weight = NA, ...)
## S3 method for class 'evonet'
as.network(x, directed = TRUE, ...)
## S3 method for class 'evonet'
as.igraph(x, directed = TRUE, use.labels = TRUE, ...)

as.evonet(x, ...)
## S3 method for class 'phylo'
as.evonet(x, ...)

read.evonet(file = "", text = NULL, comment.char = "", ...)
write.evonet(x, file = "", ...)
```

Arguments

phy	an object of class "phylo".
x	an object of class "evonet".
from	a vector (or a matrix if to = NULL) giving the node or tip numbers involved in the reticulations.
to	a vector of the same length than from.
col, lty, lwd	colors, line type and width of the reticulations (recycled if necessary).
alpha	a value between 0 and 1 specifying the transparency of the reticulations.
arrows, arrow.type	see fancyarrows.
order, index.only	see reorder.phylo.
weight	a numeric vector giving the weights for the reticulations when converting to the class "networx" (recycled or shortened if needed).
directed	a logical: should the network be considered as directed? TRUE by default.
use.labels	a logical specifying whether to use the tip and node labels when building the network of class "igraph".
file, text, comment.char	see read.tree.
...	arguments passed to other methods.

Details

evonet is a constructor function that checks the arguments.

The classes "networx", "network", and "igraph" are defined in the packages **phangorn**, **network**, and **igraph**, respectively.

read.evonet reads networks from files in extended newick format (Cardona et al. 2008).

Value

an object of class `c("evonet", "phylo")` which is made of an object of class "phylo" plus an element `reticulation` coding additional edges among nodes and uses the same coding rules than the edge matrix.

The conversion functions return an object of the appropriate class.

Author(s)

Emmanuel Paradis, Klaus Schliep

References

Cardona, G., Rossell, F., and Valiente, G. (2008) Extended Newick: it is time for a standard representation of phylogenetic networks. *BMC Bioinformatics*, **9**, 532.

See Also

`as.networx` in package **phangorn**

Examples

```
tr <- rcoal(5)
(x <- evonet(tr, 6:7, 8:9))
plot(x)
## simple example of extended Newick format:
(enet <- read.evonet(text = "((a:2,(b:1)#H1:1):1,(#H1,c:1):2);"))
plot(enet, arrows=1)
## from Fig. 2 in Cardona et al. 2008:
z <- read.evonet(text =
"((1,((2,(3,(4)Y#H1)g)e,(((Y#H1, 5)h,6)f)X#H2)c)a,((X#H2,7)d,8)b)r;")
z
plot(z)
## Not run:
if (require(igraph)) {
    plot(as.igraph(z))
}
## End(Not run)
```

---

ewLasso *Incomplete distances and edge weights of unrooted topology*

Description

This function implements a method for checking whether an incomplete set of distances satisfy certain conditions that might make it uniquely determine the edge weights of a given topology, T. It prints information about whether the graph with vertex set the set of leaves, denoted by X, and edge set the set of non-missing distance pairs, denoted by L, is connected or strongly non-bipartite. It then also checks whether L is a triplet cover for T.

Usage

```
ewLasso(X, phy)
```

Arguments

X	a distance matrix.
phy	an unrooted tree of class "phylo".

Details

Missing values must be represented by either NA or a negative value.

This implements a method for checking whether an incomplete set of distances satisfies certain conditions that might make it uniquely determine the edge weights of a given topology, T. It prints information about whether the graph, G, with vertex set the set of leaves, denoted by X, and edge set the set of non-missing distance pairs, denoted by L, is connected or strongly non-bipartite. It also checks whether L is a triplet cover for T. If G is not connected, then T does not need to be the only topology satisfying the input incomplete distances. If G is not strongly non-bipartite then the edge-weights of the edges of T are not the unique ones for which the input distance is satisfied. If L is a triplet cover, then the input distance matrix uniquely determines the edge weights of T. See Dress et al. (2012) for details.

Value

NULL, the results are printed in the console.

Author(s)

Andrei Popescu

References

Dress, A. W. M., Huber, K. T., and Steel, M. (2012) 'Lassoing' a phylogentic tree I: basic properties, shellings and covers. *Journal of Mathematical Biology*, **65(1)**, 77–105.

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, spr = TRUE, tbr = FALSE)
fastme.ols(X, nni = TRUE)
```

Arguments

X	a distance matrix; may be an object of class "dist".
nni	a logical value; TRUE to perform NNIs (default).
spr	ditto for SPRs.
tbr	ignored (see details).

Details

The code to perform topology searches based on TBR (tree bisection and reconnection) did not run correctly and has been removed after the release of **ape** 5.3. A warning is issued if tbr = TRUE.

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**, 687–705.

See Also

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

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[lower.tri(M)] <- x
M <- t(M)
M[lower.tri(M)] <- x
dimnames(M) <- list(1:8, 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

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

`getAnnotationsGenBank` *Read Annotations from GenBank*

Description

This function connects to the GenBank database and reads sequence annotations using accession number(s) given as argument.

Usage

```
getAnnotationsGenBank(access.nb, quiet = TRUE)
```

Arguments

`access.nb`	a vector of mode character giving the accession numbers.
`quiet`	a logical value indicating whether to show the progress of the downloads.

Details

The sequence annotations (a.k.a. feature list) are returned in a data frame with five or six columns: start, end, type, product, others, and gene (the last being optional). This is the same information that can be downloaded from NCBI's Web interface by clicking on 'Send to:', 'File', and then selecting 'Feature Table' under 'Format'.

A warning is given if some features are incomplete (this information is then dropped from the returned object).

A warning is given if some accession numbers are not found on GenBank.

Value

One of the followings: (i) a data frame if `access.nb` contains a single accession number; (ii) a list of data frames if `access.nb` contains several accession numbers, the names are set with `access.nb` (if some accession numbers are not found on GenBank, the corresponding entries are set to `NULL`); (iii) `NULL` if all accession numbers are not found on GenBank.

Author(s)

Emmanuel Paradis

References

https://www.ncbi.nlm.nih.gov/Sequin/table.html (Note: it seems this URL is broken; 2022-01-03)

See Also

`read.GenBank`, `read.gff`, `DNAbin`

Examples

```
## The 8 sequences of tanagers (Ramphocelus):
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:
annot.rampho <- getAnnotationsGenBank(ref)
annot.rampho
## check all annotations are the same:
unique(do.call(rbind, annot.rampho)[, -5])

## End(Not run)
```

`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.

See Also

`coalescent.intervals`, `collapsed.intervals`

howmanytrees *Calculate Numbers of Phylogenetic Trees*

Description

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

LargeNumber is a utility function to compute (approximately) large numbers from the power a^b.

Usage

```
howmanytrees(n, rooted = TRUE, binary = TRUE,
             labeled = TRUE, detail = FALSE)
LargeNumber(a, b)
## S3 method for class 'LargeNumber'
print(x, latex = FALSE, digits = 1, ...)
```

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).
a, b	two numbers.
x	an object of class "LargeNumber".
latex	a logical value specifying whether to print the number in LaTeX code in addition to return it.
digits	the number of digits printed for the real part of the large number (unused if latex = FALSE).
...	(unused).

Details

In the cases of labeled binary trees, the calculation is done directly and a single numeric value is returned (or an object of class "LargeNumber").

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. Only for the cases of rooted, binary, labeled topologies an approximate number is returned in the form a `"LargeNumber"` object.

Value

a single numeric value, an object of class `"LargeNumber"`, or in the case where `detail = TRUE` is used, a named vector or matrix.

Author(s)

Emmanuel Paradis

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, quiet = 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.
quiet	a logical controlling whether to print a message inviting the user to click on the tree.
...	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

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)
```

image.DNAbin *Plot of DNA Sequence Alignement*

Description

This function plots an image of an alignment of nucleotide sequences.

Usage

```
## S3 method for class 'DNAbin'
image(x, what, col, bg = "white", xlab = "", ylab = "",
      show.labels = TRUE, cex.lab = 1, legend = TRUE,
      grid = FALSE, show.bases = FALSE, base.cex = 1,
      base.font = 1, base.col = "black", scheme = "Ape_NT", ...)
```

Arguments

x	a matrix of DNA sequences (class "DNAbin").
what	a vector of characters specifying the bases to visualize. If missing, this is set to "a", "g", "c", "t", "n", and "-" (in this order).
col	a vector of colours. If missing, this is set to "red", "yellow", "green", "blue", "grey", and "black". If it is shorter (or longer) than what, it is recycled (or shortened).
bg	the colour used for nucleotides whose base is not among what.
xlab	the label for the *x*-axis; none by default.
ylab	Idem for the *y*-axis. Note that by default, the labels of the sequences are printed on the *y*-axis (see next option).
show.labels	a logical controlling whether the sequence labels are printed (TRUE by default).
cex.lab	a single numeric controlling the size of the sequence labels. Use cex.axis to control the size of the annotations on the *x*-axis.
legend	a logical controlling whether the legend is plotted (TRUE by default).
grid	a logical controlling whether to draw a grid (FALSE by default).
show.bases	a logical controlling whether to show the base symbols (FALSE by default).
base.cex, base.font, base.col	control the aspect of the base symbols (ignored if the previous is FALSE).
scheme	a predefined color scheme. For amino acid options are "Ape_AA", "Zappo_AA", "Clustal", "Polarity" and "Transmembrane_tendency", for nucleotides "Ape_NT" and "RY_NT".
...	further arguments passed to image.default (e.g., xlab, cex.axis).

Details

The idea of this function is to allow flexible plotting and colouring of a nucleotide alignment. By default, the most common bases (a, g, c, t, and n) and alignment gap are plotted using a standard colour scheme.

It is possible to plot only one base specified as `what` with a chosen colour: this might be useful to check, for instance, the distribution of alignment gaps (`image(x, "-")`) or missing data (see examples).

Author(s)

Emmanuel Paradis, Klaus Schliep

See Also

`DNAbin`, `del.gaps`, `alex`, `alview`, `all.equal.DNAbin`, `clustal`, `grid`, `image.AAbin`

Examples

```
data(woodmouse)
image(woodmouse)
rug(seg.sites(woodmouse), -0.02, 3, 1)
image(woodmouse, "n", "blue") # show missing data
image(woodmouse, c("g", "c"), "green") # G+C
par(mfcol = c(2, 2))
### barcoding style:
for (x in c("a", "g", "c", "t"))
    image(woodmouse, x, "black", cex.lab = 0.5, cex.axis = 0.7)
par(mfcol = c(1, 1))
### zoom on a portion of the data:
image(woodmouse[11:15, 1:50], c("a", "n"), c("blue", "grey"))
grid(50, 5, col = "black")
### see the guanines on a black background:
image(woodmouse, "g", "yellow", "black")
### Amino acid
X <- trans(woodmouse, 2)
image(X) # default ape colors
image(X, scheme="Clustal") # Clustal coloring
```

`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 <dutheil@evolbio.mpg.de>

See Also

corClasses, Initialize.corStruct.

---

is.binary *Test for Binary Tree*

Description

This function tests whether a phylogenetic tree is binary.

Usage

```
is.binary(phy)
## S3 method for class 'phylo'
is.binary(phy)
## S3 method for class 'multiPhylo'
is.binary(phy)
## S3 method for class 'tree'
is.binary(phy)
```

Arguments

phy	an object of class "phylo" or "multiPhylo".

Details

The test differs whether the tree is rooted or not. An urooted tree is considered binary if all its nodes are of degree three (i.e., three edges connect to each node). A rooted tree is considered binary if all nodes (including the root node) have exactly two descendant nodes, so that they are of degree three expect the root which is of degree 2.

The test ignores branch lengths. Consider using `di2multi` if you want to treat zero-branch lengths as resulting from multichotomies.

`is.binary.tree` is deprecated and will be removed soon: currently it calls `is.binary`.

Value

a logical vector.

Author(s)

Emmanuel Paradis

See Also

`is.rooted`, `is.ultrametric`, `multi2di`

Examples

```
is.binary(rtree(10))
is.binary(rtree(10, rooted = FALSE))
is.binary(stree(10))
x <- setNames(rmtree(10, 10), LETTERS[1:10])
is.binary(x)
```

---

is.compatible *Check Compatibility of Splits*

Description

`is.compatible` is a generic function with a method for the class `"bitsplits"`. It checks whether a set of splits is compatible using the `arecompatible` function.

Usage

```
is.compatible(obj)
## S3 method for class 'bitsplits'
is.compatible(obj)
arecompatible(x, y, n)
```

Arguments

obj	an object of class "bitsplits".
x, y	a vector of mode raw.
n	the number of taxa in the splits.

Value

TRUE if the splits are compatible, FALSE otherwise.

Author(s)

Andrei Popescu

See Also

as.bitsplits

---

is.monophyletic *Is Group Monophyletic*

Description

This function tests whether a list of tip labels is monophyletic on a given tree.

Usage

```
is.monophyletic(phy, tips, reroot = !is.rooted(phy), plot = FALSE, ...)
```

Arguments

phy	a phylogenetic tree description of class "phylo".
tips	a vector of mode numeric or character specifying the tips to be tested.
reroot	a logical. If FALSE, then the input tree is not unrooted before the test.
plot	a logical. If TRUE, then the tree is plotted with the specified group tips highlighted.
...	further arguments passed to plot.

Details

If phy is rooted, the test is done on the rooted tree, otherwise the tree is first unrooted, then arbitrarily rerooted, in order to be independent on the current position of the root. That is, the test asks if tips could be monophyletic given any favourably rooting of phy.

If phy is unrooted the test is done on an unrooted tree, unless reroot = FALSE is specified.

If tip labels in the list tips are given as characters, they need to be spelled as in the object phy.

Value

TRUE or FALSE.

Author(s)

Johan Nylander <jnylander@users.sourceforge.net>

See Also

which.edge, drop.tip, mrca.

Examples

```
## Test one monophyletic and one paraphyletic group on the bird.orders tree
## Not run: data("bird.orders")
## Not run: is.monophyletic(phy = bird.orders, tips = c("Ciconiiformes", "Gruiformes"))
## Not run: is.monophyletic(bird.orders, c("Passeriformes", "Ciconiiformes", "Gruiformes"))
```

---

is.ultrametric *Test if a Tree is Ultrametric*

Description

This function tests whether a tree is ultrametric using the distances from each tip to the root.

Usage

```
is.ultrametric(phy, ...)
## S3 method for class 'phylo'
is.ultrametric(phy, tol = .Machine$double.eps^0.5, option = 1, ...)
## S3 method for class 'multiPhylo'
is.ultrametric(phy, tol = .Machine$double.eps^0.5, option = 1, ...)
```

Arguments

phy	an object of class "phylo" or "multiPhylo".
tol	a numeric >= 0, variation below this value are considered non-significant.
option	an integer (1 or 2; see details).
...	arguments passed among methods.

Details

The test is based on the distances from each tip to the root and a criterion: if option = 1, the criterion is the scaled range ((max - min/max)), if option = 2, the variance is used (this was the method used until ape 3.5). The default criterion is invariant to linear changes of the branch lengths.

Value

a logical vector.

Author(s)

Emmanuel Paradis

See Also

is.binary, .Machine

Examples

```
is.ultrametric(rtree(10))
is.ultrametric(rcoal(10))
```

---

kronoviz *Plot Multiple Chronograms on the Same Scale*

Description

The main argument is a list of (rooted) trees which are plotted on the same scale.

Usage

```
kronoviz(x, layout = length(x), horiz = TRUE, ...,
         direction = ifelse(horiz, "rightwards", "upwards"), side = 2)
```

Arguments

x	a list of (rooted) trees of class "phylo".
layout	an integer giving the number of trees plotted simultaneously; by default all.
horiz	a logical specifying whether the trees should be plotted rightwards (the default) or upwards.
...	further arguments passed to plot.phylo.
direction	a character string specifying the direction of the tree. Four values are possible: "rightwards" (the default), "leftwards", "upwards", and "downwards".
side	Where to put the axis, see example.

Details

The size of the individual plots is proportional to the size of the trees.

Value

NULL

Author(s)

Emmanuel Paradis, Klaus Schliep

See Also

plot.phylo

Examples

```
TR <- replicate(10, rcoal(sample(11:20, size = 1)), simplify = FALSE)
kronoviz(TR)
kronoviz(TR, side = 1)
kronoviz(TR, horiz = FALSE, type = "c", show.tip.label = FALSE)
kronoviz(TR, direction = "d", side = c(1,2))
```

---

label2table *Label Management*

Description

These functions work on a vector of character strings storing bi- or trinomial species names, typically "Genus_species_subspecies".

Usage

```
label2table(x, sep = NULL, as.is = FALSE)
stripLabel(x, species = FALSE, subsp = TRUE, sep = NULL)
abbreviateGenus(x, genus = TRUE, species = FALSE, sep = NULL)
```

Arguments

x	a vector of mode character.
sep	the separator (a single character) between the taxonomic levels (see details).
as.is	a logical specifying whether to convert characters into factors (like in read.table).
species, subsp, genus	a logical specifying whether the taxonomic level is concerned by the operation.

Details

label2table returns a data frame with three columns named "genus", "species", and "subspecies" (with NA if the level is missing).

stripLabel deletes the subspecies names from the input. If species = TRUE, the species names are also removed, thus returning only the genus names.

abbreviateGenus abbreviates the genus names keeping only the first letter. If species = TRUE, the species names are abbreviated.

By default, these functions try to guess what is the separator between the genus, species and/or subspecies names. If an underscore is present in the input, then this character is assumed to be the separator; otherwise, a space. If this does not work, you can specify sep to its appropriate value.

Value

A vector of mode character or a data frame.

Author(s)

Emmanuel Paradis

See Also

makeLabel, makeNodeLabel, mixedFontLabel, updateLabel, checkLabel

Examples

```
x <- c("Panthera_leo", "Panthera_pardus", "Panthera_onca", "Panthera_uncia",
       "Panthera_tigris_altaica", "Panthera_tigris_amoyensis")
label2table(x)
stripLabel(x)
stripLabel(x, TRUE)
abbreviateGenus(x)
abbreviateGenus(x, species = TRUE)
abbreviateGenus(x, genus = FALSE, species = TRUE)
```

---

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

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))
```

---

latag2n *Leading and Trailing Alignment Gaps to N*

Description

Substitutes leading and trailing alignment gaps in aligned sequences into N (i.e., A, C, G, or T). The gaps in the middle of the sequences are left unchanged.

Usage

```
latag2n(x)
```

Arguments

x	an object of class "DNAbin" with the aligned sequences.

Details

This function is called by others in **ape** and in **pegas**. It is documented here in case it needs to be called by other packages.

Value

an object of class "DNAbin".

Author(s)

Emmanuel Paradis

See Also

DNAbin

Examples

```
x <- as.DNAbin(matrix(c("-", "A", "G", "-", "T", "C"), 2, 3))
y <- latag2n(x)
alview(x)
alview(y)
```

lmorigin *Multiple regression through the origin*

Description

Function lmorigin computes a multiple linear regression and performs tests of significance of the equation parameters (F-test of R-square and t-tests of regression coefficients) using permutations.

The regression line can be forced through the origin. Testing the significance in that case requires a special permutation procedure. This option was developed for the analysis of independent contrasts, which requires regression through the origin. A permutation test, described by Legendre & Desdevises (2009), is needed to analyze contrasts that are not normally distributed.

Usage

```
lmorigin(formula, data, origin=TRUE, nperm=999, method=NULL, silent=FALSE)
```

Arguments

formula	A formula specifying the bivariate model, as in lm and aov.
data	A data frame containing the two variables specified in the formula.
origin	origin = TRUE (default) to compute regression through the origin; origin = FALSE to compute multiple regression with estimation of the intercept.
nperm	Number of permutations for the tests. If nperm = 0, permutation tests will not be computed. The default value is nperm = 999. For large data files, the permutation test is rather slow since the permutation procedure is not compiled.
method	method = "raw" computes t-tests of the regression coefficients by permutation of the raw data. method = "residuals" computes t-tests of the regression coefficients by permutation of the residuals of the full model. If method = NULL, permutation of the raw data is used to test the regression coefficients in regression through the origin; permutation of the residuals of the full model is used to test the regression coefficients in ordinary multiple regression.
silent	Informative messages and the time to compute the tests will not be written to the R console if silent=TRUE. Useful when the function is called by a numerical simulation function.

Details

The permutation F-test of R-square is always done by permutation of the raw data. When there is a single explanatory variable, permutation of the raw data is used for the t-test of the single regression coefficient, whatever the method chosen by the user. The rationale is found in Anderson & Legendre (1999).

The print.lmorigin function prints out the results of the parametric tests (in all cases) and the results of the permutational tests (when nperm > 0).

Value

`reg`	The regression output object produced by function `lm`.
`p.param.t.2tail`	Parametric probabilities for 2-tailed tests of the regression coefficients.
`p.param.t.1tail`	Parametric probabilities for 1-tailed tests of the regression coefficients. Each test is carried out in the direction of the sign of the coefficient.
`p.perm.t.2tail`	Permutational probabilities for 2-tailed tests of the regression coefficients.
`p.perm.t.1tail`	Permutational probabilities for 1-tailed tests of the regression coefficients. Each test is carried out in the direction of the sign of the coefficient.
`p.perm.F`	Permutational probability for the F-test of R-square.
`origin`	TRUE is regression through the origin has been computed, FALSE if multiple regression with estimation of the intercept has been used.
`nperm`	Number of permutations used in the permutation tests.
`method`	Permutation method for the t-tests of the regression coefficients: `method = "raw"` or `method = "residuals"`.
`var.names`	Vector containing the names of the variables used in the regression.
`call`	The function call.

Author(s)

Pierre Legendre, Universite de Montreal

References

Anderson, M. J. and Legendre, P. (1999) An empirical comparison of permutation methods for tests of partial regression coefficients in a linear model. *Journal of Statistical Computation and Simulation*, **62**, 271–303.

Legendre, P. and Desdevises, Y. (2009) Independent contrasts and regression through the origin. *Journal of Theoretical Biology*, **259**, 727–743.

Sokal, R. R. and Rohlf, F. J. (1995) *Biometry - The principles and practice of statistics in biological research. Third edition.* New York: W. H. Freeman.

Examples

```
## Example 1 from Sokal & Rohlf (1995) Table 16.1
## SO2 air pollution in 41 cities of the USA
data(lmorigin.ex1)
out <- lmorigin(SO2 ~ ., data=lmorigin.ex1, origin=FALSE, nperm=99)
out

## Example 2: Contrasts computed on the phylogenetic tree of Lamellodiscus
## parasites. Response variable: non-specificity index (NSI); explanatory
## variable: maximum host size. Data from Table 1 of Legendre & Desdevises
## (2009).
data(lmorigin.ex2)
```

```
out <- lmorigin(NSI ~ MaxHostSize, data=lmorigin.ex2, origin=TRUE, nperm=99)
out

## Example 3: random numbers
y <- rnorm(50)
X <- as.data.frame(matrix(rnorm(250),50,5))
out <- lmorigin(y ~ ., data=X, origin=FALSE, nperm=99)
out
```

LTT *Theoretical Lineage-Through Time Plots*

Description

This function draws the lineage-through time (LTT) plots predicted under a speciation-extinction model (aka birth-death model) with specified values of speciation and extinction rates (which may vary with time).

A prediction interval is plotted by default which requires to define a sample size (100 by default), and different curves can be combined.

Usage

```
LTT(birth = 0.1, death = 0, N = 100, Tmax = 50, PI = 95,
    scaled = TRUE, eps = 0.1, add = FALSE, backward = TRUE,
    ltt.style = list("black", 1, 1), pi.style = list("blue", 1, 2), ...)
```

Arguments

`birth`	the speciation rate, this may be either a numeric value or a funtion of time (named `t` in the code of the function).
`death`	id. for the extinction rate.
`N`	the size of the tree.
`Tmax`	the age of the root of the tree.
`PI`	the percentage value of the prediction interval; set this value to 0 to not draw this interval.
`scaled`	a logical values specifying whether to scale the y-axis between 0 and 1.
`eps`	a numerical value giving the resolution of the time axis.
`add`	a logical values specifying whether to make a new plot (the default).
`backward`	a logical value: should the time axis be traced from the present (the default), or from the root of the tree?
`ltt.style`	a list with three elements giving the style of the LTT curve with, respectively, the colour (`"col"`), the line thickness (`"lwd"`), and the line type (`"lty"`).
`pi.style`	id. for the prediction interval.
`...`	arguments passed to `plot` (e.g., `log="y"`).

Details

For the moment, this works well when `birth` and `death` are constant. Some improvements are under progress for time-dependent rates (but see below for an example).

Author(s)

Emmanuel Paradis

References

Hallinan, N. (2012) The generalized time variable reconstructed birth–death process. *Journal of Theoretical Biology*, **300**, 265–276.

Paradis, E. (2011) Time-dependent speciation and extinction from phylogenies: a least squares approach. *Evolution*, **65**, 661–672.

Paradis, E. (2015) Random phylogenies and the distribution of branching times. *Journal of Theoretical Biology*, **387**, 39–45.

See Also

`ltt.plot`

Examples

```
### predicted LTT plot under a Yule model with lambda = 0.1
### and 50 species after 50 units of time...
LTT(N = 50)
### ... and with a birth-death model with the same rate of
### diversification (try with N = 500):
LTT(0.2, 0.1, N = 50, PI = 0, add = TRUE, ltt.style = list("red", 2, 1))
### predictions under different tree sizes:
layout(matrix(1:4, 2, 2, byrow = TRUE))
for (N in c(50, 100, 500, 1000)) {
    LTT(0.2, 0.1, N = N)
    title(paste("N =", N))
}
layout(1)
## Not run:
### speciation rate decreasing with time
birth.logis <- function(t) 1/(1 + exp(0.02 * t + 4))
LTT(birth.logis)
LTT(birth.logis, 0.05)
LTT(birth.logis, 0.1)

## End(Not run)
```

`ltt.plot` *Lineages Through Time Plot*

Description

These functions provide tools for plotting the numbers of lineages through time from phylogenetic trees.

Usage

```
ltt.plot(phy, xlab = "Time", ylab = "N",
         backward = TRUE, tol = 1e-6, ...)
ltt.lines(phy, backward = TRUE, tol = 1e-6, ...)
mltt.plot(phy, ..., dcol = TRUE, dlty = FALSE, legend = TRUE,
          xlab = "Time", ylab = "N", log = "", backward = TRUE,
          tol = 1e-6)
ltt.coplot(phy, backward = TRUE, ...)
ltt.plot.coords(phy, backward = TRUE, tol = 1e-6, type = "S")
```

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").
`backward`	a logical value: should the time axis be traced from the present (the default), or from the root of the tree?
`tol`	a numeric value (see details).
`...`	in the cases of `ltt.plot()`, `ltt.lines()`, or `ltt.coplot()` these are further (graphical) arguments to be passed to `plot()`, `lines()`, or `plot.phylo()`, respectively (see details on how to transform the axes); in the case of `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"`.
`type`	either `"S"` or `"s"`, the preferred type of step function, corresponding to argument `type` of base function `plot()`. See section "Value" below.

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".

The option tol is used as follows: first the most distant tip from the root is found, then all tips whose distance to the root is not different from the previous one more than tol are considered to be contemporaneous with it.

If the tree is not ultrametric, the plot is done assuming the tips, except the most distant from the root, represent extinction events. If a root edge is present, it is taken into account.

ltt.lines adds a LTT curve to an existing plot. Additional arguments (...) may be used to change the settings of the added line.

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) and/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. The plot will certainly make sense if all trees have their most-distant-from-the-root tips contemporaneous (i.e., trees with only extinct lineages will not be represented properly). For more flexible settings of line drawings, it may be better to combine ltt.plot() with successive calls of ltt.lines() (see examples).

ltt.coplot is meant to show how to set a tree and a LTT plots on the same scales. All extra arguments modify only the appearance of the tree. The code can be easily edited and tailored.

Value

ltt.plot.coords returns a two-column matrix with the time points and the number of lineages, respectively. type = "S" returns the number of lineages to the left of (or "up to") the corresponding point in time, while type = "s" returns the number of lineages to the right of this point (i.e, between that time and the next).

Author(s)

Emmanuel Paradis

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

kronoviz, skyline, LTT, branching.times, birthdeath, bd.ext, yule.cov, bd.time; plot for the basic plotting function in R

Examples

```
data(bird.families)
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
data(bird.orders)
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(1)

### better with ltt.coplot():
ltt.coplot(bird.families, show.tip.label = FALSE, x.lim = 27.5)
data(chiroptera)
chiroptera <- compute.brlen(chiroptera)
ltt.coplot(chiroptera, show.tip.label = FALSE, type = "c")

### with extinct lineages and a root edge:
omar <- par("mar")
set.seed(31)
tr <- rlineage(0.2, 0.15)
tr$root.edge <- 5
ltt.coplot(tr, show.tip.label = FALSE, x.lim = 55)
## compare with:
## ltt.coplot(drop.fossil(tr), show.tip.label = FALSE)
layout(1)
par(mar = omar)

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(-20, 10, lwd = c(2, 1), lty = c(1, 2), bty = "n",
       legend = c("Bird orders", "Random (coalescent) trees"))
```

makeLabel *Label Management*

Description

This is a generic function with methods for character vectors, trees of class "phylo", lists of trees of class "multiPhylo", and DNA sequences of class "DNAbin". All options for the class character may be used in the other methods.

Usage

```
makeLabel(x, ...)
## S3 method for class 'character'
makeLabel(x, len = 99, space = "_", make.unique = TRUE,
          illegal = "():;,[]", quote = FALSE, ...)
## S3 method for class 'phylo'
makeLabel(x, tips = TRUE, nodes = TRUE, ...)
## S3 method for class 'multiPhylo'
makeLabel(x, tips = TRUE, nodes = TRUE, ...)
## S3 method for class 'DNAbin'
makeLabel(x, ...)
```

Arguments

x	a vector of mode character or an object for which labels are to be changed.
len	the maximum length of the labels: those longer than 'len' will be truncated.
space	the character to replace spaces, tabulations, and linebreaks.
make.unique	a logical specifying whether duplicate labels should be made unique by appending numerals; TRUE by default.
illegal	a string specifying the characters to be deleted.
quote	a logical specifying whether to quote the labels; FALSE by default.
tips	a logical specifying whether tip labels are to be modified; TRUE by default.
nodes	a logical specifying whether node labels are to be modified; TRUE by default.
...	further arguments to be passed to or from other methods.

Details

The option make.unique does not work exactly in the same way then the function of the same name: numbers are suffixed to all labels that are identical (without separator). See the examples.

If there are 10–99 identical labels, the labels returned are "xxx01", "xxx02", etc, or "xxx001", "xxx002", etc, if they are 100–999, and so on. The number of digits added preserves the option 'len'.

The default for 'len' makes labels short enough to be read by PhyML. Clustal accepts labels up to 30 character long.

Value

An object of the appropriate class.

Note

The current version does not perform well when trying to make very short unique labels (e.g., less than 5 character long).

Author(s)

Emmanuel Paradis

See Also

makeNodeLabel, make.unique, make.names, abbreviate, mixedFontLabel, label2table, updateLabel, checkLabel

Examples

```
x <- rep("a", 3)
makeLabel(x)
make.unique(x) # <- from R's base
x <- rep("aaaaa", 2)
makeLabel(x, len = 3) # made unique and of length 3
makeLabel(x, len = 3, make.unique = FALSE)
```

---

makeNodeLabel *Makes Node Labels*

Description

This function makes node labels in a tree in a flexible way.

Usage

```
makeNodeLabel(phy, ...)
## S3 method for class 'phylo'
makeNodeLabel(phy, method = "number",
                      prefix = "Node", nodeList = list(), ...)
## S3 method for class 'multiPhylo'
makeNodeLabel(phy, method = "number",
                      prefix = "Node", nodeList = list(), ...)
```

Arguments

phy	an object of class "phylo".
method	a character string giving the method used to create the labels. Three choices are possible: "number" (the default), "md5sum", and "user", or any unambiguous abbreviation of these.
prefix	the prefix used if method = "number".
nodeList	a named list specifying how nodes are names if method = "user" (see details and examples).
...	further arguments passed to grep.

Details

The three methods are described below:

- "number"! The labels are created with 1, 2, ... prefixed with the argument prefix; thus the default is to have Node1, Node2, ... Set prefix = "" to have only numbers.
- "md5sum": For each node, the labels of the tips descendant from this node are extracted, sorted alphabetically, and written into a temporary file, then the md5sum of this file is extracted and used as label. This results in a 32-character string which is unique (even accross trees) for a given set of tip labels.
- "user": the argument nodeList must be a list with names, the latter will be used as node labels. For each element of nodeList, the tip labels of the tree are searched for patterns present in this element: this is done using grep. Then the most recent common ancestor of the matching tips is given the corresponding names as labels. This is repeated for each element of nodeList.

The method "user" can be used in combination with either of the two others (see examples). Note that this method only modifies the specified node labels (so that if the other nodes have already labels they are not modified) while the two others change all labels.

Value

an object of class "phylo".

Author(s)

Emmanuel Paradis

See Also

makeLabel, grep, mixedFontLabel, label2table, checkLabel

Examples

```
tr <-
"((Pan_paniscus,Pan_troglodytes),((Homo_sapiens,Homo_erectus),Homo_abilis));"
tr <- read.tree(text = tr)
tr <- makeNodeLabel(tr, "u", nodeList = list(Pan = "Pan", Homo = "Homo"))
```

```
plot(tr, show.node.label = TRUE)
### does not erase the previous node labels:
tr <- makeNodeLabel(tr, "u", nodeList = list(Hominid = c("Pan","Homo")))
plot(tr, show.node.label = TRUE)
### the two previous commands could be combined:
L <- list(Pan = "Pan", Homo = "Homo", Hominid = c("Pan","Homo"))
tr <- makeNodeLabel(tr, "u", nodeList = L)
### combining different methods:
tr <- makeNodeLabel(tr, c("n", "u"), prefix = "#", nodeList = list(Hominid = c("Pan","Homo")))
plot(tr, show.node.label = TRUE)
```

`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 second matrix randomly and calculates a Z-statistic.

Usage

```
mantel.test(m1, m2, nperm = 999, graph = FALSE,
            alternative = "two.sided",  ...)
```

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).
`alternative`	a character string defining the alternative hypothesis: `"two.sided"` (default), `"less"`, `"greater"`, or any unambiguous abbreviation of these.
`...`	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.

The present implementation can analyse symmetric as well as (since version 5.1 of **ape**) asymmetric matrices (see Mantel 1967, Sects. 4 and 5). The diagonals of both matrices are ignored.

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).
alternative	the alternative hypothesis.

Author(s)

Original code in S by Ben Bolker, ported to R by Julien Claude

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)
diag(q1) <- diag(q2) <- 0
mantel.test(q1, q2, graph = TRUE,
            main = "Mantel test: a random example with 6 X 6 matrices
representing asymmetric relationships",
            xlab = "z-statistic", ylab = "Density",
            sub = "The vertical line shows the observed z-statistic")
```

---

mat3 *Three Matrices*

Description

Three matrices respectively representing Serological (asymmetric), DNA hybridization (asymmetric) and Anatomical (symmetric) distances among 9 families.

Usage

```
data(mat3)
```

Format

A data frame with 27 observations and 9 variables.

Source

Lapointe, F.-J., J. A. W. Kirsch and J. M. Hutcheon. 1999. Total evidence, consensus, and bat phylogeny: a distance-based approach. Molecular Phylogenetics and Evolution 11: 55-66.

See Also

mat5Mrand, mat5M3ID

---

mat5M3ID *Five Trees*

Description

Three partly similar trees, two independent trees.

Usage

```
data(mat5M3ID)
```

Format

A data frame with 250 observations and 50 variables.

Source

Data provided by V. Campbell.

See Also

mat5Mrand, mat3

---

mat5Mrand *Five Independent Trees*

Description

Five independent additive trees.

Usage

```
data(mat5Mrand)
```

Format

A data frame with 250 observations and 50 variables.

Source

Data provided by V. Campbell.

See Also

mat5M3ID, mat3

---

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

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))
```

`mcconwaysims.test` *McConway-Sims Test of Homogeneous Diversification*

Description

This function performs the McConway–Sims test that a trait or variable does not affect diversification rate.

Usage

```
mcconwaysims.test(x)
```

Arguments

`x`	a matrix or a data frame with at least two columns: the first one gives the number of species in clades with a trait supposed to increase or decrease diversification rate, and the second one the number of species in the sister-clades without the trait. Each row represents a pair of sister-clades.

Details

The McConway–Sims test compares a series of sister-clades where one of the two is characterized by a trait supposed to affect diversification rate. The null hypothesis is that the trait does not affect diversification. The alternative hypothesis is that diversification rate is increased or decreased by the trait (by contrast to the Slowinski–Guyer test). The test is a likelihood-ratio of a null Yule model and an alternative model with two parameters.

Value

a data frame with the χ^2, the number of degrees of freedom, and the *P*-value.

Author(s)

Emmanuel Paradis

References

McConway, K. J. and Sims, H. J. (2004) A likelihood-based method for testing for nonstochastic variation of diversification rates in phylogenies. *Evolution*, **58**, 12–23.

Paradis, E. (2012) Shift in diversification in sister-clade comparisons: a more powerful test. *Evolution*, **66**, 288–295.

See Also

`balance`, `slowinskiguyer.test`, `rc` in **geiger**, `shift.test` in **apTreeshape**

Examples

```
### simulate 10 clades with lambda = 0.1 and mu = 0.09:
n0 <- replicate(10, balance(rbdtree(.1, .09, Tmax = 35))[1])
### simulate 10 clades with lambda = 0.15 and mu = 0.1:
n1 <- replicate(10, balance(rbdtree(.15, .1, Tmax = 35))[1])
x <- cbind(n1, n0)
mcconwaysims.test(x)
slowinskiguyer.test(x)
richness.yule.test(x, 35)
```

`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.in=0)
## S3 method for class 'popsize'
plot(x, show.median=TRUE, show.years=FALSE,
             subst.rate, present.year, xlab = NULL,
             ylab = "Effective population size", log = "y", ...)
## 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).

`xlab` label on the x-axis (depends on the value of `show.years`).

`ylab` label on the y-axis.

`log` log-transformation of axes; by default, the y-axis is log-transformed.

`...` Further arguments to be passed on to `plot` or `lines`.

Details

Please refer to Opgen-Rhein et al. (2005) for methodological details, and the help page of skyline for information on a related approach.

Author(s)

Rainer Opgen-Rhein and Korbinian Strimmer. Parts of the rjMCMC sampling procedure are adapted from R code by Karl Broman.

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) # toy run
#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)
```

---

mixedFontLabel *Mixed Font Labels for Plotting*

Description

This function helps to format labels with bits of text in different font shapes (italics, bold, or bolditalics) and different separators. The output is intended to be used for plotting.

Usage

```
mixedFontLabel(..., sep = " ", italic = NULL, bold = NULL,
               parenthesis = NULL,
               always.upright = c("sp.", "spp.", "ssp."))
```

Arguments

...	vectors of mode character to be formatted. They may be of different lengths in which case the shortest ones are recycled.
sep	a vector of mode character giving the separators to be printed between the elements in ....
italic	a vector of integers specifying the elements in ... to be printed in italics.
bold	id. in boldface.
parenthesis	id. within parentheses.
always.upright	of vector of mode character giving the strings to not print in italics. Use always.upright = "" to cancel this option.

Details

The idea is to have different bits of text in different vectors that are put together to make a vector of R expressions. This vector is interpreted by graphical functions to format the text. A simple use may be mixedFontLabel(genus, species, italic = 1:2), but it is more interesting when mixing fonts (see examples).

To have an element in bolditalics, its number must given in both italic and bold.

The vector returned by this function may be assigned as the tip.label element of a tree of class "phylo", or even as its node.label element.

Value

A vector of mode expression.

Author(s)

Emmanuel Paradis

See Also

makeLabel, makeNodeLabel, label2table, updateLabel, checkLabel

Examples

```
tr <- read.tree(text = "((a,(b,c)),d);")
genus <- c("Gorilla", "Pan", "Homo", "Pongo")
species <- c("gorilla", "spp.", "sapiens", "pygmaeus")
geo <- c("Africa", "Africa", "World", "Asia")
tr$tip.label <- mixedFontLabel(genus, species, geo, italic = 1:2,
  parenthesis = 3)
layout(matrix(c(1, 2), 2))
plot(tr)
tr$tip.label <- mixedFontLabel(genus, species, geo, sep = c(" ", " | "),
  italic = 1:2, bold = 3)
plot(tr)
layout(1)
```

`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 <dutheil@evolbio.mpg.de> and Emmanuel Paradis

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
```

MPR *Most Parsimonious Reconstruction*

Description

This function does ancestral character reconstruction by parsimony as described in Hanazawa et al. (1995) and modified by Narushima and Hanazawa (1997).

Usage

```
MPR(x, phy, outgroup)
```

Arguments

x	a vector of integers.
phy	an object of class "phylo"; the tree must be unrooted and fully dichotomous.
outgroup	an integer or a character string giving the tip of phy used as outgroup.

Details

Hanazawa et al. (1995) and Narushima and Hanazawa (1997) used Farris's (1970) and Swofford and Maddison's (1987) framework to reconstruct ancestral states using parsimony. The character is assumed to take integer values. The algorithm finds the sets of values for each node as intervals with lower and upper values.

It is recommended to root the tree with the outgroup before the analysis, so plotting the values with nodelabels is simple.

Value

a matrix of integers with two columns named "lower" and "upper" giving the lower and upper values of the reconstructed sets for each node.

Author(s)

Emmanuel Paradis

References

Farris, J. M. (1970) Methods for computing Wagner trees. *Systematic Zoology*, **19**, 83–92.

Hanazawa, M., Narushima, H. and Minaka, N. (1995) Generating most parsimonious reconstructions on a tree: a generalization of the Farris–Swofford–Maddison method. *Discrete Applied Mathematics*, **56**, 245–265.

Narushima, H. and Hanazawa, M. (1997) A more efficient algorithm for MPR problems in phylogeny. *Discrete Applied Mathematics*, **80**, 231–238.

Swofford, D. L. and Maddison, W. P. (1987) Reconstructing ancestral character states under Wagner parsimony. *Mathematical Biosciences*, **87**, 199–229.

See Also

ace, root, nodelabels

Examples

```
## the example in Narushima and Hanazawa (1997):
tr <- read.tree(text = "(((i,j)c,(k,l)b)a,(h,g)e,f)d;")
x <- c(1, 3, 0, 6, 5, 2, 4)
names(x) <- letters[6:12]
(o <- MPR(x, tr, "f"))
plot(tr)
nodelabels(paste0("[", o[, 1], ",", o[, 2], "]"))
tiplabels(x[tr$tip.label], adj = -2)
```

```
## some random data:
x <- rpois(30, 1)
tr <- rtree(30, rooted = FALSE)
MPR(x, tr, "t1")
```

---

mrca *Find Most Recent Common Ancestors Between Pairs*

Description

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

getMRCA returns the MRCA of two or more tips.

Usage

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

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.
tip	a vector of mode numeric or character specifying the tips; can also be node numbers.

Details

For mrca, 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.

For getMRCA, if tip is of length one or zero then NULL is returned.

Value

a matrix of mode numeric (mrca) or a single numeric value (getMRCA).

Author(s)

Emmanuel Paradis, Klaus Schliep, Joseph W. Brown

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 <julien.claude@umontpellier.fr> and Emmanuel Paradis

See Also

dist.dna, dist.gene, dist, plot

Examples

```
require(stats)
X <- matrix(runif(200), 20, 10)
d <- dist(X)
PC <- prcomp(X)
M <- mst(d)
opar <- 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, ...)
## S3 method for class 'phylo'
multi2di(phy, random = TRUE, equiprob = TRUE, ...)
## S3 method for class 'multiPhylo'
multi2di(phy, random = TRUE, equiprob = TRUE, ...)
di2multi(phy, ...)
## S3 method for class 'phylo'
di2multi(phy, tol = 1e-08, ...)
## S3 method for class 'multiPhylo'
di2multi(phy, tol = 1e-08, ...)
```

Arguments

phy	an object of class "phylo" or "multiPhylo".
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).
equiprob	a logical value: should topologies generated in equal probabilities; see details in rtree (ignored if random = FALSE).
tol	a numeric value giving the tolerance to consider a branch length significantly greater than zero.
...	arguments passed among methods.

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

an object of the same class than the input.

Author(s)

Emmanuel Paradis

See Also

is.binary

Examples

```
data(bird.families)
is.binary(bird.families)
is.binary(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(1)
```

multiphylo *Manipulating Lists of Trees*

Description

These are extraction and replacement operators for lists of trees stored in the class "multiPhylo".

Usage

```
## S3 method for class 'multiPhylo'
x[i]
## S3 method for class 'multiPhylo'
x[[i]]
## S3 method for class 'multiPhylo'
x$name
## S3 replacement method for class 'multiPhylo'
x[i] <- value
## S3 replacement method for class 'multiPhylo'
x[[i]] <- value
## S3 replacement method for class 'multiPhylo'
x$i <- value
```

Arguments

x, value	an object of class "phylo" or "multiPhylo".
i	index(ices) 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.

Details

The subsetting operator [keeps the class correctly ("multiPhylo").

The replacement operators check the labels of value if x has a single vector of tip labels for all trees (see examples).

Value

An object of class "phylo" ([[, $) or of class "multiPhylo" ([and the replacement operators).

Author(s)

Emmanuel Paradis

See Also

summary.phylo, c.phylo

Examples

```
x <- rmtree(10, 20)
names(x) <- paste("tree", 1:10, sep = "")
x[1:5]
x[1] # subsetting
x[[1]] # extraction
x$tree1 # same than above
x[[1]] <- rtree(20)

y <- .compressTipLabel(x)
## up to here 'x' and 'y' have exactly the same information
## but 'y' has a unique vector of tip labels for all the trees
x[[1]] <- rtree(10) # no error
try(y[[1]] <- rtree(10)) # error

try(x[1] <- rtree(20)) # error
## use instead one of the two:
x[1] <- list(rtree(20))
x[1] <- c(rtree(20))

x[1:5] <- rmtree(5, 20) # replacement
x[11:20] <- rmtree(10, 20) # elongation
x # 20 trees
```

---

mvr *Minimum Variance Reduction*

Description

Phylogenetic tree construction based on the minimum variance reduction.

Usage

```
mvr(X, V)
mvrs(X, V, fs = 15)
```

Arguments

X	a distance matrix.
V	a variance matrix.
fs	agglomeration criterion parameter: it is coerced as an integer and must at least equal to one.

Details

The MVR method can be seen as a version of BIONJ which is not restricted to the Poison model of variance (Gascuel 2000).

Value

an object of class "phylo".

Author(s)

Andrei Popescu

References

Criscuolo, A. and Gascuel, O. (2008). Fast NJ-like algorithms to deal with incomplete distance matrices. *BMC Bioinformatics*, 9.

Gascuel, O. (2000). Data model and classification by trees: the minimum variance reduction (MVR) method. *Journal of Classification*, **17**, 67–99.

See Also

bionj, fastme, njs, SDM

Examples

```
data(woodmouse)
rt <- dist.dna(woodmouse, variance = TRUE)
v <- attr(rt, "variance")
tr <- mvr(rt, v)
plot(tr, "u")
```

---

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

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.

Studier, J. A. and Keppler, K. J. (1988) A note on the neighbor-joining algorithm of Saitou and Nei. *Molecular Biology and Evolution*, **5**, 729–731.

See Also

write.tree, read.tree, dist.dna, bionj, fastme, njs

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[lower.tri(M)] <- x
M <- t(M)
M[lower.tri(M)] <- x
dimnames(M) <- list(1:8, 1:8)
tr <- nj(M)
plot(tr, "u")
### a less theoretical example
data(woodmouse)
trw <- nj(dist.dna(woodmouse))
plot(trw)
```

---

njs *Tree Reconstruction from Incomplete Distances With NJ* or bio-NJ**

Description

Reconstructs a phylogenetic tree from a distance matrix with possibly missing values.

Usage

```
njs(X, fs = 15)
bionjs(X, fs = 15)
```

Arguments

X	a distance matrix.
fs	argument *s* of the agglomerative criterion: it is coerced as an integer and must at least equal to one.

Details

Missing values represented by either NA or any negative number.

Basically, the Q* criterion is applied to all the pairs of leaves, and the *s* highest scoring ones are chosen for further analysis by the agglomeration criteria that better handle missing distances (see references for details).

Value

an object of class "phylo".

Author(s)

Andrei Popescu

References

Criscuolo, A., Gascuel, O. (2008) Fast NJ-like algorithms to deal with incomplete distance matrices. *BMC Bioinformatics*, **9**, 166.

See Also

nj, bionj, triangMtds

Examples

```
data(woodmouse)
d <- dist.dna(woodmouse)
dm <- d
dm[sample(length(dm), size = 3)] <- NA
dist.topo(njs(dm), nj(d)) # often 0
dm[sample(length(dm), size = 10)] <- NA
dist.topo(njs(dm), nj(d)) # sometimes 0
```

---

node.dating *node.dating*

Description

Estimate the dates of a rooted phylogenetic tree from the tip dates.

Usage

```
estimate.mu(t, node.dates, p.tol = 0.05)
estimate.dates(t, node.dates, mu = estimate.mu(t, node.dates),
               min.date = -.Machine$double.xmax, show.steps = 0,
               opt.tol = 1e-8, nsteps = 1000,
               lik.tol = 0, is.binary = is.binary.phylo(t))
```

Arguments

t	an object of class "phylo"
node.dates	a numeric vector of dates for the tips, in the same order as 't$tip.label' or a vector of dates for all of the nodes.
p.tol	p-value cutoff for failed regression.
mu	mutation rate.
min.date	the minimum bound on the dates of nodes
show.steps	print the log likelihood every show.steps. If 0 will supress output.
opt.tol	tolerance for optimization precision.
lik.tol	tolerance for likelihood comparison.
nsteps	the maximum number of steps to run.
is.binary	if TRUE, will run a faster optimization method that only works if the tree is binary; otherwise will use optimize() as the optimization method.

Details

This code duplicates the functionality of the program Tip.Dates (see references). The dates of the internal nodes of 't' are estimated using a maximum likelihood approach.

't' must be rooted and have branch lengths in units of expected substitutions per site.

'node.dates' can be either a numeric vector of dates for the tips or a numeric vector for all of the nodes of 't'. 'estimate.mu' will use all of the values given in 'node.dates' to estimate the mutation rate. Dates can be censored with NA. 'node.dates' must contain all of the tip dates when it is a parameter of 'estimate.dates'. If only tip dates are given, then 'estimate.dates' will run an initial step to estimate the dates of the internal nodes. If 'node.dates' contains dates for some of the nodes, 'estimate.dates' will use those dates as priors in the inital step. If all of the dates for nodes are given, then 'estimate.dates' will not run the inital step.

If 'is.binary' is set to FALSE, 'estimate.dates' uses the "optimize" function as the optimization method. By default, R's "optimize" function uses a precision of ".Machine$double.eps^0.25", which is about 0.0001 on a 64-bit system. This should be set to a smaller value if the branch lengths of 't' are very short. If 'is.binary' is set to TRUE, estimate dates uses calculus to deterimine the maximum likelihood at each step, which is faster. The bounds of permissible values are reduced by 'opt.tol'.

'estimate.dates' has several criteria to decide how many steps it will run. If 'lik.tol' and 'nsteps' are both 0, then 'estimate.dates' will only run the initial step. If 'lik.tol' is greater than 0 and 'nsteps' is 0, then 'estimate.dates' will run until the difference between successive steps is less than 'lik.tol'. If 'lik.tol' is 0 and 'nsteps' is greater than 0, then 'estimate.dates' will run the inital step and then 'nsteps' steps. If 'lik.tol' and 'nsteps' are both greater than 0, then 'estimate.dates' will run the inital step and then either 'nsteps' steps or until the difference between successive steps is less than 'lik.tol'.

Value

The estimated mutation rate as a numeric vector of length one for estimate.mu.

The estimated dates of all of the nodes of the tree as a numeric vector with length equal to the number of nodes in the tree.

Note

This model assumes that the tree follows a molecular clock. It only performs a rudimentary statistical test of the molecular clock hypothesis.

Author(s)

Bradley R. Jones <email: brj1@sfu.ca>

References

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

Rambaut, A. (2000) Estimating the rate of molecular evolution: incorporating non-contemporaneous sequences into maximum likelihood phylogenies. *Bioinformatics*, **16**, 395–399.

Jones, Bradley R., and Poon, Art F. Y. (2016) node.dating: dating ancestors in phylogenetic trees in R *Bioinformatics*, **33**, 932–934.

See Also

optimize, rtt, plotTreeTime

Examples

```
t <- rtree(100)
tip.date <- rnorm(t$tip.label, mean = node.depth.edgelength(t)[1:Ntip(t)])^2
t <- rtt(t, tip.date)
mu <- estimate.mu(t, tip.date)

## Run for 100 steps
node.date <- estimate.dates(t, tip.date, mu, nsteps = 100)

## Run until the difference between successive log likelihoods is
## less than $10^{-4}$ starting with the 100th step's results
node.date <- estimate.dates(t, node.date, mu, nsteps = 0, lik.tol = 1e-4)

## To rescale the tree over time
t$edge.length <- node.date[t$edge[, 2]] - node.date[t$edge[, 1]]
```

---

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

Description

These functions return the depths or heights of nodes and tips.

Usage

```
node.depth(phy, method = 1)
node.depth.edgelength(phy)
node.height(phy, clado.style = FALSE)
```

Arguments

phy	an object of class "phylo".
method	an integer value (1 or 2); 1: the node depths are proportional to the number of tips descending from each node, 2: they are evenly spaced.
clado.style	a logical value; if TRUE, the node heights are calculated for a cladogram.

Details

node.depth computes the depth of a node depending on the value of method (see the option node.depth in plot.phylo). The value of 1 is given to the tips.

node.depth.edgelength does the same but using branch lengths.

node.height computes the heights of nodes and tips as plotted by a phylogram or a cladogram.

Value

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

Author(s)

Emmanuel Paradis

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", horiz = FALSE,
           width = NULL, height = NULL, ...)
tiplabels(text, tip, adj = c(0.5, 0.5), frame = "rect",
          pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
          col = "black", bg = "yellow", horiz = FALSE,
          width = NULL, height = NULL, offset = 0, ...)
edgelabels(text, edge, adj = c(0.5, 0.5), frame = "rect",
           pch = NULL, thermo = NULL, pie = NULL, piecol = NULL,
           col = "black", bg = "lightgreen", horiz = FALSE,
           width = NULL, height = NULL, date = NULL, ...)
```

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 or symbols. 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). It can be a data frame which is then converted into a matrix.
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).

`horiz`, `width`, `height`
: parameters controlling the aspect of thermometers; by default, their width and height are determined automatically.

`offset`
: offset of the tip labels (can be negative).

`date`
: specifies the positions of labels on edges of chronograms with respect to the time scale.

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, Ben Bolker, and Jim Lemon

See Also

`plot.phylo`, `edges`, `mixedFontLabel`

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)

## A trick by Liam Revell when there are many categories:
plot(tr, x.lim = c(-1, 4))
nodelabels(node = 4, pie = matrix(rep(1, 100), 1), cex = 5)
op <- par(fg = "transparent")
nodelabels(node = 5, pie = matrix(rep(1, 100), 1), cex = 5)
par(op)

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$tip.label[i], i, adj = 0)
### Some random data to compare piecharts and thermometres:
tr <- rtree(15)
x <- runif(14, 0, 0.33)
y <- runif(14, 0, 0.33)
z <- runif(14, 0, 0.33)
x <- cbind(x, y, z, 1 - x - y - z)
layout(matrix(1:2, 1, 2))
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(pie = x, cex = 1.3)
text(4.5, 15, "Are you \"pie\"...", font = 4, cex = 1.5)
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(thermo = x, col = rainbow(4), cex = 1.3)
text(4.5, 15, "... or \"thermo\"?", font = 4, cex = 1.5)
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(thermo = x, col = rainbow(4), cex = 1.3)
plot(tr, "c", FALSE, no.margin = TRUE)
nodelabels(thermo = x, col = rainbow(4), width = 3, horiz = TRUE)
layout(1)
plot(tr, main = "Showing Edge Lengths")
edgelabels(round(tr$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")
```

nodepath *Find Paths of Nodes*

Description

This function finds paths of nodes in a tree. The nodes can be internal and/or terminal (i.e., tips).

Usage

```
nodepath(phy, from = NULL, to = NULL)
```

Arguments

phy	an object of class "phylo".
from, to	integers giving node or tip numbers.

Details

By default, this function returns all the paths from the root to each tip of the tree. If both arguments from and to are specified, the shortest path of nodes linking them is returned.

Value

a list of vectors of integers (by default), or a single vector of integers.

Author(s)

Emmanuel Paradis

See Also

getMRCA

Examples

```
tr <- rtree(2)
nodepath(tr)
nodepath(tr, 1, 2)
```

parafit *Test of host-parasite coevolution*

Description

Function parafit tests the hypothesis of coevolution between a clade of hosts and a clade of parasites. The null hypothesis (H0) of the global test is that the evolution of the two groups, as revealed by the two phylogenetic trees and the set of host-parasite association links, has been independent. Tests of individual host-parasite links are also available as an option.

The method, which is described in detail in Legendre et al. (2002), requires some estimates of the phylogenetic trees or phylogenetic distances, and also a description of the host-parasite associations (H-P links) observed in nature.

Usage

```
parafit(host.D, para.D, HP, nperm = 999, test.links = FALSE,
        seed = NULL, correction = "none", silent = FALSE)
```

Arguments

host.D	A matrix of phylogenetic or patristic distances among the hosts (object class: matrix, data.frame or dist). A matrix of patristic distances exactly represents the information in a phylogenetic tree.
para.D	A matrix of phylogenetic or patristic distances among the parasites (object class: matrix, data.frame or dist). A matrix of patristic distances exactly represents the information in a phylogenetic tree.
HP	A rectangular matrix with hosts as rows and parasites as columns. The matrix contains 1's when a host-parasite link has been observed in nature between the host in the row and the parasite in the column, and 0's otherwise.
nperm	Number of permutations for the tests. If nperm = 0, permutation tests will not be computed. The default value is nperm = 999. For large data files, the permutation test is rather slow since the permutation procedure is not compiled.
test.links	test.links = TRUE will test the significance of individual host-parasite links. Default: test.links = FALSE.
seed	seed = NULL (default): a seed is chosen at random by the function. That seed is used as the starting point for all tests of significance, i.e. the global H-P test and the tests of individual H-P links if they are requested. Users can select a seed of their choice by giving any integer value to seed, for example seed = -123456. Running the function again with the same seed value will produce the exact same test results.
correction	Correction methods for negative eigenvalues (details below): correction="lingoes" and correction="cailliez". Default value: "none".
silent	Informative messages and the time to compute the tests will not be written to the R console if silent=TRUE. Useful when the function is called by a numerical simulation function.

Details

Two types of test are produced by the program: a global test of coevolution and, optionally, a test on the individual host-parasite (H-P) link.

The function computes principal coordinates for the host and the parasite distance matrices. The principal coordinates (all of them) act as a complete representation of either the phylogenetic distance matrix or the phylogenetic tree.

Phylogenetic distance matrices are normally Euclidean. Patristic distance matrices are additive, thus they are metric and Euclidean. Euclidean matrices are fully represented by real-valued principal coordinate axes. For non-Euclidean matrices, negative eigenvalues are produced; complex principal coordinate axes are associated with the negative eigenvalues. So, the program rejects matrices that are not Euclidean and stops.

Negative eigenvalues can be corrected for by one of two methods: the Lingoes or the Caillez correction. It is up to the user to decide which correction method should be applied. This is done by selecting the option `correction="lingoes"` or `correction="cailliez"`. Details on these correction methods are given in the help file of the `pcoa` function.

The principle of the global test is the following (H0: independent evolution of the hosts and parasites): (1) Compute matrix D = C t(A) B. Note: D is a fourth-corner matrix (sensu Legendre et al. 1997), where A is the H-P link matrix, B is the matrix of principal coordinates computed from the host.D matrix, and C is the matrix of principal coordinates computed from the para.D matrix. (2) Compute the statistic ParaFitGlobal, the sum of squares of all values in matrix D. (3) Permute at random, separately, each row of matrix A, obtaining matrix A.perm. Compute D.perm = C

The test of each individual H-P link is carried out as follows (H0: this particular link is random): (1) Remove one link (k) from matrix A. (2) Compute matrix D = C t(A) B. (3a) Compute trace(k), the sum of squares of all values in matrix D. (3b) Compute the statistic ParaFitLink1 = (trace - trace(k)) where trace is the ParaFitGlobal statistic. (3c) Compute the statistic ParaFitLink2 = (trace - trace(k)) / (tracemax - trace) where tracemax is the maximum value that can be taken by trace. (4) Permute at random, separately, each row of matrix A, obtaining A.perm. Use the same sequences of permutations as were used in the test of ParaFitGlobal. Using the values of trace and trace.perm saved during the global test, compute the permuted values of the two statistics, ParaFit1.perm and ParaFit2.perm. (5) Repeat step 4 a large number of times. (6) Add the reference value of ParaFit1 to the distribution of ParaFit1.perm values; add the reference value of ParaFit2 to the distribution of ParaFit2.perm values. Calculate the permutational probabilities associated to ParaFit1 and ParaFit2.

The `print.parafit` function prints out the results of the global test and, optionally, the results of the tests of the individual host-parasite links.

Value

`ParaFitGlobal`	The statistic of the global H-P test.
`p.global`	The permutational p-value associated with the ParaFitGlobal statistic.
`link.table`	The results of the tests of individual H-P links, including the ParaFitLink1 and ParaFitLink2 statistics and the p-values obtained from their respective permutational tests.
`para.per.host`	Number of parasites per host.
`host.per.para`	Number of hosts per parasite.
`nperm`	Number of permutations for the tests.

Author(s)

Pierre Legendre, Universite de Montreal

References

Hafner, M. S, P. D. Sudman, F. X. Villablanca, T. A. Spradling, J. W. Demastes and S. A. Nadler. 1994. Disparate rates of molecular evolution in cospeciating hosts and parasites. *Science*, **265**, 1087–1090.

Legendre, P., Y. Desdevises and E. Bazin. 2002. A statistical test for host-parasite coevolution. *Systematic Biology*, **51(2)**, 217–234.

See Also

pcoa

Examples

```
## Gopher and lice data from Hafner et al. (1994)

data(gopher.D)
data(lice.D)
data(HP.links)

res <- parafit(gopher.D, lice.D, HP.links, nperm=99, test.links=TRUE)
# res     # or else: print(res)
```

---

pcoa *Principal Coordinate Analysis*

Description

Function pcoa computes principal coordinate decomposition (also called classical scaling) of a distance matrix D (Gower 1966). It implements two correction methods for negative eigenvalues.

Usage

```
pcoa(D, correction="none", rn=NULL)

## S3 method for class 'pcoa'
biplot(x, Y=NULL, plot.axes = c(1,2), dir.axis1=1,
       dir.axis2=1, rn=NULL, main=NULL, ...)
```

Arguments

D	A distance matrix of class dist or matrix.
correction	Correction methods for negative eigenvalues (details below): "lingoes" and "cailliez". Default value: "none".
rn	An optional vector of row names, of length n, for the n objects.
x	Output object from pcoa.
Y	Any rectangular data table containing explanatory variables to be projected onto the ordination plot. That table may contain, for example, the community composition data used to compute D, or any transformation of these data; see examples.
plot.axes	The two PCoA axes to plot.
dir.axis1	= -1 to revert axis 1 for the projection of points and variables. Default value: +1.
dir.axis2	= -1 to revert axis 2 for the projection of points and variables. Default value: +1.
main	An optional title.
...	Other graphical arguments passed to function.

Details

This function implements two methods for correcting for negative values in principal coordinate analysis (PCoA). Negative eigenvalues can be produced in PCoA when decomposing distance matrices produced by coefficients that are not Euclidean (Gower and Legendre 1986,Legendre and Legendre 1998).

In pcoa, when negative eigenvalues are present in the decomposition results, the distance matrix D can be modified using either the Lingoes or the Cailliez procedure to produce results without negative eigenvalues.

In the Lingoes (1971) procedure, a constant c1, equal to twice absolute value of the largest negative value of the original principal coordinate analysis, is added to each original squared distance in the distance matrix, except the diagonal values. A newe principal coordinate analysis, performed on the modified distances, has at most (n-2) positive eigenvalues, at least 2 null eigenvalues, and no negative eigenvalue.

In the Cailliez (1983) procedure, a constant c2 is added to the original distances in the distance matrix, except the diagonal values. The calculation of c2 is described in Legendre and Legendre (1998). A new principal coordinate analysis, performed on the modified distances, has at most (n-2) positive eigenvalues, at least 2 null eigenvalues, and no negative eigenvalue.

In all cases, only the eigenvectors corresponding to positive eigenvalues are shown in the output list. The eigenvectors are scaled to the square root of the corresponding eigenvalues. Gower (1966) has shown that eigenvectors scaled in that way preserve the original distance (in the D matrix) among the objects. These eigenvectors can be used to plot ordination graphs of the objects.

We recommend not to use PCoA to produce ordinations from the chord, chi-square, abundance profile, or Hellinger distances. It is easier to first transform the community composition data using the following transformations, available in the decostand function of the vegan package, and then carry out a principal component analysis (PCA) on the transformed data:

- Chord transformation: decostand(spiders,"normalize")
- Transformation to relative abundance profiles: decostand(spiders,"total")

- Hellinger transformation: decostand(spiders,"hellinger")
- Chi-square transformation: decostand(spiders,"chi.square")

The ordination results will be identical and the calculations shorter. This two-step ordination method, called transformation-based PCA (tb-PCA), was described by Legendre and Gallagher (2001).

The `biplot.pcoa` function produces plots for any pair of principal coordinates. The original variables can be projected onto the ordination plot.

Value

`correction`	The values of parameter `correction` and variable 'correct' in the function.
`note`	A note describing the type of correction done, if any.
`values`	The eigenvalues and related information:
`Eigenvalues`	All eigenvalues (positive, null, negative).
`Relative_eig`	Relative eigenvalues.
`Corr_eig`	Corrected eigenvalues (Lingoes correction); Legendre and Legendre (1998, p. 438, eq. 9.27).
`Rel_corr_eig`	Relative eigenvalues after Lingoes or Cailliez correction.
`Broken_stick`	Expected fractions of variance under the broken stick model.
`Cumul_eig`	Cumulative relative eigenvalues.
`Cum_corr_eig`	Cumulative corrected relative eigenvalues.
`Cumul_br_stick`	Cumulative broken stick fractions.
`vectors`	The principal coordinates with positive eigenvalues.
`trace`	The trace of the distance matrix. This is also the sum of all eigenvalues, positive and negative.
`vectors.cor`	The principal coordinates with positive eigenvalues from the distance matrix corrected using the method specified by parameter `correction`.
`trace.cor`	The trace of the corrected distance matrix. This is also the sum of its eigenvalues.

Author(s)

Pierre Legendre, Universite de Montreal

References

Cailliez, F. (1983) The analytical solution of the additive constant problem. *Psychometrika*, **48**, 305–308.

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

Gower, J. C. and Legendre, P. (1986) Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification*, **3**, 5–48.

Legendre, P. and Gallagher, E. D. (2001) Ecologically meaningful transformations for ordination of species data. *Oecologia*, **129**, 271–280.

Legendre, P. and Legendre, L. (1998) *Numerical Ecology, 2nd English edition.* Amsterdam: Elsevier Science BV.

Lingoes, J. C. (1971) Some boundary conditions for a monotone analysis of symmetric matrices. *Psychometrika*, **36**, 195–203.

Examples

```
## Oribatid mite data from Borcard and Legendre (1994)

## Not run:
if (require(vegan)) {
data(mite) # Community composition data, 70 peat cores, 35 species

## Select rows 1:30. Species 35 is absent from these rows. Transform to log
mite.log <- log(mite[1:30, -35] + 1)  # Equivalent: log1p(mite[1:30, -35])

## Principal coordinate analysis and simple ordination plot
mite.D <- vegdist(mite.log, "bray")
res <- pcoa(mite.D)
res$values
biplot(res)

## Project unstandardized and standardized species on the PCoA ordination plot
mite.log.st = apply(mite.log, 2, scale, center=TRUE, scale=TRUE)

par(mfrow=c(1,2))
biplot(res, mite.log)
biplot(res, mite.log.st)

# Reverse the ordination axes in the  plot
par(mfrow=c(1,2))
biplot(res, mite.log, dir.axis1=-1, dir.axis2=-1)
biplot(res, mite.log.st, dir.axis1=-1, dir.axis2=-1)
}
## End(Not run)
```

`phydataplot` *Tree Annotation*

Description

`phydataplot` plots data on a tree in a way that adapts to the type of tree. `ring` does the same for circular trees.

Both functions match the data with the labels of the tree.

Usage

```
phydataplot(x, phy, style = "bars", offset = 1, scaling = 1,
            continuous = FALSE, width = NULL, legend = "below",
            funcol = rainbow, ...)
ring(x, phy, style = "ring", offset = 1, ...)
```

Arguments

x	a vector, a factor, a matrix, or a data frame.
phy	the tree (which must be already plotted).
style	a character string specifying the type of graphics; can be abbreviated (see details).
offset	the space between the tips of the tree and the plot.
scaling	the scaling factor to apply to the data.
continuous	(used if style="mosaic") a logical specifying whether to treat the values in x as continuous or not; can be an integer value giving the number of categories.
width	(used if style = "mosaic") the width of the cells; by default, all the available space is used.
legend	(used if style = "mosaic") the place where to draw the legend; one of "below" (the default), "side", or "none", or an unambiguous abbreviation of these.
funcol	(used if style = "mosaic") the function used to generate the colours (see details and examples).
...	further arguments passed to the graphical functions.

Details

The possible values for style are "bars", "segments", "image", "arrows", "boxplot", "dotchart", or "mosaic" for phydataplot, and "ring", "segments", or "arrows" for ring.

style = "image" works only with square matrices (e.g., similarities). If you want to plot a DNA alignment in the same way than image.DNAbin, try style = "mosaic".

style = "mosaic" can plot any kind of matrices, possibly after discretizing its values (using continuous). The default colour palette is taken from the function rainbow. If you want to use specified colours, a function simply returning the vector of colours must be used, possibly with names if you want to assign a specific colour to each value (see examples).

Note

For the moment, only rightwards trees are supported (does not apply to circular trees).

Author(s)

Emmanuel Paradis

See Also

plot.phylo, nodelabels, fancyarrows

Examples

```
## demonstrates matching with names:
tr <- rcoal(n <- 10)
x <- 1:n
names(x) <- tr$tip.label
plot(tr, x.lim = 11)
phydataplot(x, tr)
## shuffle x but matching names with tip labels reorders them:
phydataplot(sample(x), tr, "s", lwd = 3, lty = 3)

## adapts to the tree:
plot(tr, "f", x.l = c(-11, 11), y.l = c(-11, 11))
phydataplot(x, tr, "s")

## leave more space with x.lim to show a barplot and a dotchart:
plot(tr, x.lim = 22)
phydataplot(x, tr, col = "yellow")
phydataplot(x, tr, "d", offset = 13)

ts <- rcoal(N <- 100)
X <- rTraitCont(ts) # names are set
dd <- dist(X)
op <- par(mar = rep(0, 4))
plot(ts, x.lim = 10, cex = 0.4, font = 1)
phydataplot(as.matrix(dd), ts, "i", offset = 0.2)

par(xpd = TRUE, mar = op$mar)
co <- c("blue", "red"); l <- c(-2, 2)
X <- X + abs(min(X)) # move scale so X >= 0
plot(ts, "f", show.tip.label = FALSE, x.lim = l, y.lim = l, open.angle = 30)
phydataplot(X, ts, "s", col = co, offset = 0.05)
ring(X, ts, "ring", col = co, offset = max(X) + 0.1) # the same info as a ring

## as many rings as you want...
co <- c("blue", "yellow")
plot(ts, "r", show.tip.label = FALSE, x.l = c(-1, 1), y.l = c(-1, 1))
for (o in seq(0, 0.4, 0.2)) {
    co <- rev(co)
    ring(0.2, ts, "r", col = rep(co, each = 5), offset = o)
}

lim <- c(-5, 5)
co <- rgb(0, 0.4, 1, alpha = 0.1)
y <- seq(0.01, 1, 0.01)
plot(ts, "f", x.lim = lim, y.lim = lim, show.tip.label = FALSE)
ring(y, ts, offset = 0, col = co, lwd = 0.1)
for (i in 1:3) {
    y <- y + 1
    ring(y, ts, offset = 0, col = co, lwd = 0.1)
}

## rings can be in the background
```

```
plot(ts, "r", plot = FALSE)
ring(1, ts, "r", col = rainbow(100), offset = -1)
par(new = TRUE)
plot(ts, "r", font = 1, edge.color = "white")

## might be more useful:
co <- c("lightblue", "yellow")
plot(ts, "r", plot = FALSE)
ring(0.1, ts, "r", col = sample(co, size = N, rep = TRUE), offset = -.1)
par(new = TRUE)
plot(ts, "r", font = 1)

## if x is matrix:
tx <- rcoal(m <- 20)
X <- runif(m, 0, 0.5); Y <- runif(m, 0, 0.5)
X <- cbind(X, Y, 1 - X - Y)
rownames(X) <- tx$tip.label
plot(tx, x.lim = 6)
co <- rgb(diag(3))
phydataplot(X, tx, col = co)
## a variation:
plot(tx, show.tip.label = FALSE, x.lim = 5)
phydataplot(X, tx, col = co, offset = 0.05, border = NA)

plot(tx, "f", show.tip.label = FALSE, open.angle = 180)
ring(X, tx, col = co, offset = 0.05)

Z <- matrix(rnorm(m * 5), m)
rownames(Z) <- rownames(X)
plot(tx, x.lim = 5)
phydataplot(Z, tx, "bo", scaling = .5, offset = 0.5,
            boxfill = c("gold", "skyblue"))

## plot an alignment with a NJ tree:
data(woodmouse)
trw <- nj(dist.dna(woodmouse))
plot(trw, x.lim = 0.1, align.tip = TRUE, font = 1)
phydataplot(woodmouse[, 1:50], trw, "m", 0.02, border = NA)

## use type = "mosaic" on a 30x5 matrix:
tr <- rtree(n <- 30)
p <- 5
x <- matrix(sample(3, size = n*p, replace = TRUE), n, p)
dimnames(x) <- list(paste0("t", 1:n), LETTERS[1:p])
plot(tr, x.lim = 35, align.tip = TRUE, adj = 1)
phydataplot(x, tr, "m", 2)
## change the aspect:
plot(tr, x.lim = 35, align.tip = TRUE, adj = 1)
phydataplot(x, tr, "m", 2, width = 2, border = "white", lwd = 3, legend = "side")
## user-defined colour:
f <- function(n) c("yellow", "blue", "red")
phydataplot(x, tr, "m", 18, width = 2, border = "white", lwd = 3,
            legend = "side", funcol = f)
```

```
## alternative colour function...:
## fb <- function(n) c("3" = "red", "2" = "blue", "1" = "yellow")
## ... but since the values are sorted alphabetically,
## both f and fb will produce the same plot.

## use continuous = TRUE with two different scales:
x[] <- 1:(n*p)
plot(tr, x.lim = 35, align.tip = TRUE, adj = 1)
phydataplot(x, tr, "m", 2, width = 1.5, continuous = TRUE, legend = "side",
            funcol = colorRampPalette(c("white", "darkgreen")))
phydataplot(x, tr, "m", 18, width = 1.5, continuous = 5, legend = "side",
            funcol = topo.colors)
```

`phymltest` *Fits a Bunch of Models with PhyML*

Description

This function calls 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 = NULL, append = TRUE)
## 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 executable. This argument can be left as NULL if PhyML's default names are used: "phyml_3.0_linux32", "phyml_3.0_macintel", or "phyml_3.0_win32.exe", under Linux, MacOS, or Windows respectively.
append	a logical indicating whether to erase previous PhyML output files if present; the default is to not erase.
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 requires version 3.0.1 of PhyML; it won't work with older versions.

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.

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 simply 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).

The option path2exec has been removed in the present version: the path to PhyML's executable can be specified with the option execname.

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

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://www.atgc-montpellier.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) <- ape:::.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,
    rescaled.tree = FALSE)
```

Arguments

x	a numeric vector.
phy	an object of class "phylo".
scaled	logical, indicates whether the contrasts should be scaled with their expected variances (default to TRUE).
var.contrasts	logical, indicates whether the expected variances of the contrasts should be returned (default to FALSE).
rescaled.tree	logical, if TRUE the rescaled tree is returned together with the main results.

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. 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). If the tree has node labels, these are used as labels of the returned object.

If rescaled.tree = TRUE, a list is returned with two elements named "contr" with the above results and "rescaled.tree" with the tree and its rescaled branch lengths (see Felsenstein 1985).

Author(s)

Emmanuel Paradis

References

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

See Also

read.tree, compar.gee, compar.lynch, pic.ortho, varCompPhylip

Examples

```
### The example in Phylip 3.5c (originally from Lynch 1991)
x <- "((((Homo:0.21,Pongo:0.21):0.28,Macaca:0.49):0.13,Ateles:0.62):0.38,Galago:1.00);"
tree.primates <- read.tree(text = x)
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
```

---

pic.ortho *Phylogenetically Independent Orthonormal Contrasts*

Description

This function computes the orthonormal contrasts using the method described by Felsenstein (2008). Only a single trait can be analyzed; there can be several observations per species.

Usage

```
pic.ortho(x, phy, var.contrasts = FALSE, intra = FALSE)
```

Arguments

x	a numeric vector or a list of numeric vectors.
phy	an object of class "phylo".
var.contrasts	logical, indicates whether the expected variances of the contrasts should be returned (default to FALSE).
intra	logical, whether to return the intraspecific contrasts.

Details

The data x can be in two forms: a vector if there is a single observation for each species, or a list whose elements are vectors containing the individual observations for each species. These vectors may be of different lengths.

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.

Value

either a vector of contrasts, or a two-column matrix with the contrasts in the first column and their expected variances in the second column (if `var.contrasts = TRUE`). If the tree has node labels, these are used as labels of the returned object.

If `intra = TRUE`, the attribute `"intra"`, a list of vectors with the intraspecific contrasts or `NULL` for the species with a one observation, is attached to the returned object.

Author(s)

Emmanuel Paradis

References

Felsenstein, J. (2008) Comparative methods with sampling error and within-species variation: Contrasts revisited and revised. *American Naturalist*, **171**, 713–725.

See Also

`pic`, `varCompPhylip`

Examples

```
tr <- rcoal(30)
### a single observation per species:
x <- rTraitCont(tr)
pic.ortho(x, tr)
pic.ortho(x, tr, TRUE)
### different number of observations per species:
x <- lapply(sample(1:5, 30, TRUE), rnorm)
pic.ortho(x, tr, intra = TRUE)
```

---

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

See Also

correlogram.formula, Moran.I

---

plot.phylo *Plot Phylogenies*

Description

These functions plot phylogenetic trees.

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 = NULL, edge.width
     = NULL, edge.lty = NULL, node.color = NULL, node.width
     = NULL, node.lty = NULL, 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 =
     NULL, tip.color = par("col"), plot = TRUE, rotate.tree
     = 0, open.angle = 0, node.depth = 1, align.tip.label =
     FALSE, ...)
## 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", "tidy", 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.
edge.lty	same as the previous argument but for line types; 1: plain, 2: dashed, 3: dotted, 4: dotdash, 5: longdash, 6: twodash.
node.color	a vector of mode character giving the colours used to draw the perpendicular lines associated with each node of the plotted phylogeny. These are taken to be in the same order than the component node of phy. If fewer colours are given than the length of node, then the colours are recycled.

node.width	as the previous argument, but for line widths.
node.lty	as the previous argument, but for line types; 1: plain, 2: dashed, 3: dotted, 4: dotdash, 5: longdash, 6: twodash.
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 (can be abbreviated): either "horizontal" where all labels are horizontal (the default if type = "u"), or "axial" where the labels are displayed in the axis of the corresponding terminal branches. This option has an effect if type = "u", "f", or "r".
tip.color	the colours used for the tip labels, eventually recycled (see examples).
plot	a logical controlling whether to draw the tree. If FALSE, the graphical device is set as if the tree was plotted, and the coordinates are saved as well.
rotate.tree	for "fan", "unrooted", or "radial" trees: the rotation of the whole tree in degrees (negative values are accepted).
open.angle	if type = "f" or "r", the angle in degrees left blank. Use a non-zero value if you want to call axisPhylo after the tree is plotted.

node.depth — an integer value (1 or 2) used if branch lengths are not used to plot the tree; 1: the node depths are proportional to the number of tips descending from each node (the default and was the only possibility previously), 2: they are evenly spaced.

align.tip.label — a logical value or an integer. If TRUE, the tips are aligned and dotted lines are drawn between the tips of the tree and the labels. If an integer, the tips are aligned and this gives the type of the lines (lty).

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 use align.tip.label = TRUE with type = "fan", you will have certainly to set x.lim and y.lim manually.

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

node.depth

show.tip.label
show.node.label

font
cex
adj
srt
no.margin
label.offset
x.lim
y.lim
direction
tip.color
Ntip
Nnode
root.time
align.tip.label

Note

The argument asp cannot be passed with ....

Author(s)

Emmanuel Paradis, Martin Smith, Damien de Vienne

References

van der Ploeg, A. (2014) Drawing non-layered tidy trees in linear time. *Journal of Software: Practice and Experience*, **44**, 1467–1484.

See Also

read.tree, trex, kronoviz, add.scale.bar, axisPhylo, nodelabels, edges, plot for the basic plotting function in R

Examples

```
### An extract from Sibley and Ahlquist (1990)
x <- "(((Strix_aluco:4.2,Asio_otus:4.2):3.1,Athene_noctua:7.3):6.3,Tyto_alba:13.5);"
tree.owls <- read.tree(text= x)
plot(tree.owls)

### 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(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)
```

```
## tidy trees
tr <- rtree(100)
layout(matrix(1:2, 2))
plot(tr)
axis(2)
plot(tr, "t")
axis(2)
## around 20 percent gain on the y-axis
```

`plot.phylo.extra` *Extra Fuctions to Plot and Annotate Phylogenies*

Description

These are extra functions to plot and annotate phylogenies, mostly calling basic graphical functions in **ape**.

Usage

```
plotBreakLongEdges(phy, n = 1, ...)
drawSupportOnEdges(value, ...)
```

Arguments

`phy`	an object of class `"phylo"`.
`n`	the numner of long branches to be broken.
`value`	the values to be printed on the internal branches of the tree.
`...`	further arguments to be passed to `plot.phylo` or to `edgelabels`.

Details

`drawSupportOnEdges` assumes the tree is unrooted, so the vector `value` should have as many values than the number of internal branches (= number of nodes - 1). If there is one additional value, it is assumed that it relates to the root node and is dropped (see examples).

Value

NULL

Author(s)

Emmanuel Paradis

See Also

`plot.phylo`, `edgelabels`, `boot.phylo`, `plotTreeTime`

Examples

```
tr <- rtree(10)
tr$edge.length[c(1, 18)] <- 100
op <- par(mfcol = 1:2)
plot(tr); axisPhylo()
plotBreakLongEdges(tr, 2); axisPhylo()

## from ?boot.phylo:
f <- function(x) nj(dist.dna(x))
data(woodmouse)
tw <- f(woodmouse) # NJ tree with K80 distance
set.seed(1)
## bootstrap with 100 replications:
(bp <- boot.phylo(tw, woodmouse, f, quiet = TRUE))
## the first value relates to the root node and is always 100
## it is ignored below:
plot(tw, "u")
drawSupportOnEdges(bp)
## more readable but the tree is really unrooted:
plot(tw)
drawSupportOnEdges(bp)
par(op)
```

---

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 <dutheil@evolbio.mpg.de>

See Also

varcomp

---

plotTreeTime *Plot Tree With Time Axis*

Description

This function plots a non-ultrametric tree where the tips are not contemporary together with their dates on the x-axis.

Usage

```
plotTreeTime(phy, tip.dates, show.tip.label = FALSE, y.lim = NULL,
             color = TRUE, ...)
```

Arguments

phy	an object of class "phylo".
tip.dates	a vector of the same length than the number of tips in phy (see details).
show.tip.label	a logical value; see plot.phylo.
y.lim	by default, one fifth of the plot is left below the tree; use this option to change this behaviour.
color	a logical value specifying whether to use colors for the lines linking the tips to the time axis. If FALSE, a grey scale is used.
...	other arguments to be passed to plot.phylo.

Details

The vector tip.dates may be numeric or of class "Date". In either case, the time axis is set accordingly. The length of this vector must be equal to the number of tips of the tree: the dates are matched to the tips numbers. Missing values are allowed.

Value

NULL

Author(s)

Emmanuel Paradis

See Also

plot.phylo, estimate.dates

Examples

```
dates <- as.Date(.leap.seconds)
tr <- rtree(length(dates))
plotTreeTime(tr, dates)

## handling NA's:
dates[11:26] <- NA
plotTreeTime(tr, dates)

## dates can be on an arbitrary scale, e.g., [-1, 1]:
plotTreeTime(tr, runif(Ntip(tr), -1, 1))
```

---

print.phylo *Compact Display of a Phylogeny*

Description

These functions prints a compact summary of a phylogeny, or a list of phylogenies, on the console.

Usage

```
## S3 method for class 'phylo'
print(x, printlen = 6 ,...)
## S3 method for class 'multiPhylo'
print(x, details = FALSE ,...)
## 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.
...	further arguments passed to or from other methods.

Value

NULL.

Author(s)

Ben Bolker and Emmanuel Paradis

See Also

read.tree, summary.phylo, print for the generic R function

Examples

```
x <- rtree(10)
print(x)
print(x, printlen = 10)
x <- rmtree(2, 10)
print(x)
print(x, TRUE)
str(x)
```

---

rDNAbin *Random DNA Sequences*

Description

This function generates random sets of DNA sequences.

Usage

```
rDNAbin(n, nrow, ncol, base.freq = rep(0.25, 4), prefix = "Ind_")
```

Arguments

n	a vector of integers giving the lengths of the sequences. Can be missing in which case nrow and ncol must be given.
nrow, ncol	two single integer values giving the number of sequences and the number of sites, respectively (ignored if n is given).
base.freq	the base frequencies.
prefix	the prefix used to give labels to the sequences; by default these are Ind_1, ... Ind_n (or Ind_nrow).

Details

If n is used, this function generates a list with sequence lengths given by the values in n. If n is missing, a matrix is generated.

The purpose of this function is to generate a set of sequences of a specific size. To simulate sequences on a phylogenetic tree, see simSeq in **phangorn** (very efficient), and the package **phylosim** (more for pedagogy).

Value

an object of class "DNAbin".

Note

It is not recommended to use this function to generate objects larger than two billion bases (2 Gb).

Author(s)

Emmanuel Paradis

See Also

DNAbin

Examples

```
rDNAbin(1:10)
rDNAbin(rep(10, 10))
rDNAbin(nrow = 10, ncol = 10)
```

---

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".

Warning

The branch length support is still experimental and was not fully tested.

Author(s)

Julien Dutheil <dutheil@evolbio.mpg.de>

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)
fl <- tempfile("tree", fileext = ".tre")
cat("AAA","Strix_aluco","AAB","Asio_otus",
   "AB","Athene_noctua","B","Tyto_alba",
   file = fl, sep = "\n")
tree.owls <- read.caic(fl)
plot(tree.owls)
tree.owls
unlink(fl) # delete the file "ex.tre"
```

---

read.dna *Read DNA Sequences in a File*

Description

These functions read 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 returned in binary format, otherwise (if as.character = TRUE) in lowercase.

Usage

```
read.dna(file, format = "interleaved", skip = 0,
         nlines = 0, comment.char = "#",
         as.character = FALSE, as.matrix = NULL)
read.FASTA(file, type = "DNA")
read.fastq(file, offset = -33)
```

Arguments

file	a file name specified by either a variable of mode character, or a double-quoted string. Can also be a connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). Files compressed with GZIP can be read (the name must end with .gz), as well as remote files.
format	a character string specifying the format of the DNA sequences. Four choices are possible: "interleaved", "sequential", "clustal", or "fasta", or any unambiguous abbreviation of these.
skip	the number of lines of the input file to skip before beginning to read data (ignored for FASTA files; see below).
nlines	the number of lines to be read (by default the file is read untill its end; ignored for FASTA files)).
comment.char	a single character, the remaining of the line after this character is ignored (ignored for FASTA files).
as.character	a logical controlling whether to return the sequences as an object of class "DNAbin" (the default).
as.matrix	(used if format = "fasta") one of the three followings: (i) NULL: returns the sequences in a matrix if they are of the same length, otherwise in a list; (ii) TRUE: returns the sequences in a matrix, or stops with an error if they are of different lengths; (iii) FALSE: always returns the sequences in a list.
type	a character string giving the type of the sequences: one of "DNA" or "AA" (case-independent, can be abbreviated).
offset	the value to be added to the quality scores (the default applies to the Sanger format and should work for most recent FASTQ files).

Details

read.dna 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. 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 and FASTQ formats, the conventions defined in the 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. Particularities for each format are detailed below.

- Interleaved: the function starts to read the sequences after it finds one or more spaces (or tabulations). All characters before the sequences are taken as the taxa names after removing the leading and trailing spaces (so spaces in taxa names are not 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.

- Clustal: this is the format output by the Clustal programs (.aln). It is close to the interleaved format: the differences are that the dimensions of the data are not indicated in the file, and the names of the sequences are repeated in each block.
- FASTA: this looks like the sequential format but the taxa names (or 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 are ignored.

The FASTQ format is explained in the references.

Compressed files must be read through connections (see examples). `read.fastq` can read compressed files directly (see examples).

Value

a matrix or a list (if `format = "fasta"`) of DNA sequences stored in binary format, or of mode character (if `as.character = "TRUE"`).

`read.FASTA` always returns a list of class `"DNAbin"` or `"AAbin"`.

`read.fastq` returns a list of class `"DNAbin"` with an atrribute `"QUAL"` (see examples).

Author(s)

Emmanuel Paradis and RJ Ewing

References

Anonymous. FASTA format. https://en.wikipedia.org/wiki/FASTA_format

Anonymous. FASTQ format. https://en.wikipedia.org/wiki/FASTQ_format

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

```
## 1. Simple text files

TEXTfile <- tempfile("exdna", fileext = ".txt")

## 1a. Extract from data(woodmouse) in sequential format:
cat("3 40",
"No305     NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
"No304     ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
"No306     ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
file = TEXTfile, sep = "\n")
ex.dna <- read.dna(TEXTfile, format = "sequential")
str(ex.dna)
```

```
ex.dna

## 1b. 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 = TEXTfile, sep = "\n")
ex.dna2 <- read.dna(TEXTfile)

## 1c. ... in clustal format, ...
cat("CLUSTAL (ape) multiple sequence alignment", "",
"No305     NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
"No304     ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
"No306     ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
"           ************************* ******  ****",
file = TEXTfile, sep = "\n")
ex.dna3 <- read.dna(TEXTfile, format = "clustal")

## 1d. ... and in FASTA format
FASTAfile <- tempfile("exdna", fileext = ".fas")
cat(">No305",
"NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
">No304",
"ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
">No306",
"ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
file = FASTAfile, sep = "\n")
ex.dna4 <- read.dna(FASTAfile, format = "fasta")

## The 4 data objects are the same:
identical(ex.dna, ex.dna2)
identical(ex.dna, ex.dna3)
identical(ex.dna, ex.dna4)

## 2. How to read GZ compressed files

## create a GZ file and open a connection:
GZfile <- tempfile("exdna", fileext = ".fas.gz")
con <- gzfile(GZfile, "wt")
## write the data using the connection:
cat(">No305", "NTTCGAAAAACACACCCACTACTAAAANTTATCAGTCACT",
    ">No304", "ATTCGAAAAACACACCCACTACTAAAAATTATCAACCACT",
    ">No306", "ATTCGAAAAACACACCCACTACTAAAAATTATCAATCACT",
    file = con, sep = "\n")
close(con) # close the connection

## read the GZ'ed file:
ex.dna5 <- read.dna(gzfile(GZfile), "fasta")
```

```
## This example is with a FASTA file but this works as well
## with the other formats described above.

## All 5 data objects are identical:
identical(ex.dna, ex.dna5)

unlink(c(TEXTfile, FASTAfile, GZfile)) # clean-up

## Not run:
## 3. How to read files from a ZIP archive

## NOTE: since ape 5.7-1, all files in these examples are written
## in the temporary directory, thus the following commands work
## best when run in the user's working directory.

## write the woodmouse data in a FASTA file:
data(woodmouse)
write.dna(woodmouse, "woodmouse.fas", "fasta")
## archive a FASTA file in a ZIP file:
zip("myarchive.zip", "woodmouse.fas")
## Note: the file myarchive.zip is created if necessary

## Read the FASTA file from the ZIP archive without extraction:
wood2 <- read.dna(unz("myarchive.zip", "woodmouse.fas"), "fasta")

## Alternatively, unzip the archive:
fl <- unzip("myarchive.zip")
## the previous command eventually creates locally
## the fullpath archived with 'woodmouse.fas'
wood3 <- read.dna(fl, "fasta")

identical(woodmouse, wood2)
identical(woodmouse, wood3)

## End(Not run)

## read a FASTQ file from 1000 Genomes:
## Not run:
a <- "https://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/HG00096/sequence_read/"
file <- "SRR062641.filt.fastq.gz"
URL <- paste0(a, file)
download.file(URL, file)
## If the above command doesn't work, you may copy/paste URL in
## a Web browser instead.
X <- read.fastq(file)
X # 109,811 sequences
## get the qualities of the first sequence:
(qual1 <- attr(X, "QUAL")[[1]])
## the corresponding probabilities:
10^(-qual1/10)
## get the mean quality for each sequence:
mean.qual <- sapply(attr(X, "Q"), mean)
## can do the same for var, sd, ...
```

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

`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, chunk.size = 400, quiet = TRUE,
             type = "DNA")
```

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).
`chunk.size`	the number of sequences downloaded together (see details).
`quiet`	a logical value indicating whether to show the progress of the downloads. If `TRUE`, will also print the (full) name of the FASTA file containing the downloaded sequences.
`type`	a character specifying to download "DNA" (nucleotide) or "AA" (amino acid) sequences.

Details

The function uses the site https://www.ncbi.nlm.nih.gov/ from where the sequences are retrieved.

If `species.names = TRUE`, the returned list has an attribute `"species"` containing the names of the species taken from the field "ORGANISM" in GenBank.

Since **ape** 3.6, this function retrieves the sequences in FASTA format: this is more efficient and more flexible (scaffolds and contigs can be read) than what was done in previous versions. The option `gene.names` has been removed in **ape** 5.4; this information is also present in the description.

Setting `species.names = FALSE` is much faster (could be useful if you read a series of scaffolds or contigs, or if you already have the species names).

The argument `chunk.size` is set by default to 400 which is likely to work in many cases. If an error occurs such as "Cannot open file ..." showing the list of the accession numbers, then you may try decreasing `chunk.size` to 200 or 300.

If quiet = FALSE, the display is done chunk by chunk, so the message "Downloading sequences: 400 / 400 ..." means that the download from sequence 1 to sequence 400 is under progress (it is not possible to display a more accurate message because the download method depends on the platform).

Value

A list of DNA sequences made of vectors of class "DNAbin", or of single characters (if as.character = TRUE) with two attributes (species and description).

Author(s)

Emmanuel Paradis and Klaus Schliep

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]]
## the description from each FASTA sequence:
attr(Rampho, "description")

## End(Not run)
```

---

read.gff *Read GFF Files*

Description

This function reads a file in general feature format version 3 (GFF3) and returns a data frame.

Usage

```
read.gff(file, na.strings = c(".", "?"), GFF3 = TRUE)
```

Arguments

file	a file name specified by a character string.
na.strings	the strings in the GFF file that will be converted as NA's (missing values).
GFF3	a logical value specifying whether if the file is formatted according to version 3 of GFF.

Details

The returned data frame has its (column) names correctly set (see References) and the categorical variables (seqid, source, type, strand, and phase) set as factors.

This function should be more efficient than using `read.delim`.

GFF2 (aka GTF) files can also be read: use `GFF3 = FALSE` to have the correct field names. Note that GFF2 files and GFF3 files have the same structure, although some fields are slightly different (see reference).

The file can be gz-compressed (see examples), but not zipped.

Value

NULL

Author(s)

Emmanuel Paradis

References

https://en.wikipedia.org/wiki/General_feature_format

Examples

```
## Not run:
## requires to be connected on Internet
d <- "https://ftp.ensembl.org/pub/release-86/gff3/homo_sapiens/"
f <- "Homo_sapiens.GRCh38.86.chromosome.MT.gff3.gz"
download.file(paste0(d, f), "mt_gff3.gz")
## If the above command doesn't work, you may copy/paste the full URL in
## a Web browser instead.
gff.mito <- read.gff("mt_gff3.gz")
## the lengths of the sequence features:
gff.mito$end - (gff.mito$start - 1)
table(gff.mito$type)
## where the exons start:
gff.mito$start[gff.mito$type == "exon"]

## End(Not run)
```

`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, force.multi = FALSE)
```

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 giving names to the individual trees (by default, this uses the labels in the NEXUS file if these are present).
`force.multi`	a logical value; if `TRUE`, an object of class `"multiPhylo"` is always returned even if the file contains a single tree (see details).

Details

The present implementation tries to follow as much as possible the NEXUS standard (but see the restriction below on TRANSLATION tables). Only the block "TREES" is read; the other data can be read with other functions (e.g., `read.dna`, `read.table`, ...).

If a TRANSLATION table is present it is assumed that only the tip labels are translated and they are all translated with integers without gap. Consequently, if nodes have labels in the tree(s) they are read as they are and not looked for in the translation table. The logic behind this is that in the vast majority of cases, node labels will be support values rather than proper taxa names. This is consistent with `write.nexus` which translates only the tip labels.

Using `force.multi = TRUE` when the file contains a single tree makes possible to keep the tree name (as names of the list).

'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"` or `"multiPhylo"`.

Author(s)

Emmanuel Paradis

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

`read.nexus.data` reads a file with sequences in the NEXUS format. `nexus2DNAbin` is a helper function to convert the output from the previous function into the class `"DNAbin"`.

For the moment, only sequence data (DNA or protein) are supported.

Usage

```
read.nexus.data(file)
nexus2DNAbin(x)
```

Arguments

`file`	a file name specified by either a variable of mode character, or a double-quoted string.
`x`	an object output by `read.nexus.data`.

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, Thomas Guillerme, and Klaus Schliep

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 = "", keep.multi = FALSE, ...)
```

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()`).
`keep.multi`	if `TRUE` and `tree.names = NULL` then single trees are returned in `"multiPhylo"` format, with any name that is present (see details). Default is `FALSE`.
`...`	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.

If there are any characters preceding the first "(" in a line then this is assigned to the name. This is returned when a "multiPhylo" object is returned and `tree.names = NULL`.

Until **ape** 4.1, the default of `comment.char` was `"#"` (as in `scan`). This has been changed so that extended Newick files can be read.

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"`. The name of each tree can be specified by `tree.names`, or can be read from the file (see details).

Author(s)

Emmanuel Paradis and Daniel Lawson <dan.lawson@bristol.ac.uk>

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. (2020) Definition of Formats for Coding Phylogenetic Trees in R. https://emmanuelparadis.github.io/misc/FormatTreeR.pdf

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

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)
s <- "owls(((Strix_aluco:4.2,Asio_otus:4.2):3.1,Athene_noctua:7.3):6.3,Tyto_alba:13.5);"
treefile <- tempfile("tree", fileext = ".tre")
cat(s, file = treefile, sep = "\n")
tree.owls <- read.tree(treefile)
str(tree.owls)
tree.owls
tree.owls <- read.tree(treefile, keep.multi = TRUE)
tree.owls
names(tree.owls)
unlink(treefile) # clean-up
### 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

## tree with singleton nodes:
ts <- read.tree(text = "((((a))),d);")
plot(ts, node.depth = 2) # the default will overlap the singleton node with the tip
nodelabels()

## 'skeleton' tree with a singleton node:
tx <- read.tree(text = "(((,)),);")
plot(tx, node.depth = 2)
nodelabels()

## a tree with single quoted labels (the 2nd label is not quoted
## because it has no white spaces):
z <- "(('a: France, Spain (Europe)',b),'c: Australia [Outgroup]');"
tz <- read.tree(text = z)
plot(tz, font = 1)
```

---

reconstruct *Continuous Ancestral Character Estimation*

Description

This function estimates ancestral character states, and the associated uncertainty, for continuous characters. It mainly works as the ace function, from which it differs, first, in the fact that computations are not performed by numerical optimisation but through matrix calculus. Second, besides classical Brownian-based reconstruction methods, it reconstructs ancestral states under Arithmetic Brownian Motion (ABM, i.e. Brownian with linear trend) and Ornstein-Uhlenbeck process (OU, i.e. Brownian with an attractive optimum).

Usage

```
reconstruct(x, phyInit, method = "ML", alpha = NULL,
            low_alpha = 0.0001, up_alpha = 1, CI = TRUE)
```

Arguments

x	a numerical vector.
phyInit	an object of class "phylo".
method	a character specifying the method used for estimation. Six choices are possible: "ML", "REML", "GLS", "GLS_ABM", "GLS_OU" or "GLS_OUS".
alpha	a numerical value which accounts for the attractive strength parameter of "GLS_OU" or "GLS_OUS" (used only in these cases). If alpha = NULL (the default), then it is estimated by maximum likelihood using optim, with low_alpha (resp. up_alpha) as lower value (resp. upper value), which may lead to convergence issue.
low_alpha	a lower bound for alpha, used only with methods "GLS_OU" or "GLS_OUS". It has to be positive.
up_alpha	an upper bound for alpha, used only with methods "GLS_OU" or "GLS_OUS". It has to be positive.
CI	a logical specifying whether to return the 95% confidence intervals of the ancestral state estimates.

Details

For "ML", "REML" and "GLS", the default model is Brownian motion. This model can be fitted by maximum likelihood (method = "ML", Felsenstein 1973, Schluter et al. 1997) - the default, residual maximum likelihood (method = "REML"), or generalized least squares (method = "GLS", Martins and Hansen 1997, Garland T and Ives AR 2000). "GLS_ABM" is based on Brownian motion with trend model. Both "GLS_OU" and "GLS_OUS" are based on Ornstein-Uhlenbeck model. "GLS_OU" and "GLS_OUS" differs in the fact that "GLS_OUS" assume that the process starts from the optimum, while the root state has to be estimated for "GLS_OU", which may rise some issues (see Royer-Carenzi and Didier, 2016). Users may provide the attractive strength parameter alpha, for these two models. "GLS_ABM", "GLS_OU" and "GLS_OUS" are all fitted by generalized least squares (Royer-Carenzi and Didier, 2016).

Value

an object of class "ace" with the following elements:

ace	the estimates of the ancestral character values.
CI95	the estimated 95% confidence intervals.
sigma2	if method = "ML", the maximum likelihood estimate of the Brownian parameter.
loglik	if method = "ML", the maximum log-likelihood.

Note

GLS_ABM should not be used on ultrametric tree.

GLS_OU may lead to aberrant reconstructions.

Author(s)

Manuela Royer-Carenzi, Gilles Didier

References

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

Garland T. 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.

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.

Royer-Carenzi, M. and Didier, G. (2016) A comparison of ancestral state reconstruction methods for quantitative characters. *Journal of Theoretical Biology*, **404**, 126–142.

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

Yang, Z. (2006) *Computational Molecular Evolution*. Oxford: Oxford University Press.

See Also

MPR, corBrownian, compar.ou

Reconstruction of ancestral sequences can be done with the package **phangorn** (see function ?ancestral.pml).

Examples

```
### Some random data...
data(bird.orders)
x <- rnorm(23, m=100)
### Reconstruct ancestral quantitative characters:
reconstruct(x, bird.orders)
reconstruct(x, bird.orders, method = "GLS_OUS", alpha=NULL)
```

---

reorder.phylo *Internal Reordering of Trees*

Description

reorder 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.

cladewise and postorder are convenience functions to return only the indices of the reordered edge matrices (see examples).

Usage

```
## S3 method for class 'phylo'
reorder(x, order = "cladewise", index.only = FALSE, ...)
## S3 method for class 'multiPhylo'
reorder(x, order = "cladewise", ...)
cladewise(x)
postorder(x)
```

Arguments

x	an object of class "phylo" or "multiPhylo".
order	a character string: either "cladewise" (the default), "postorder", "pruningwise", or any unambiguous abbreviation of these.
index.only	should the function return only the ordered indices of the rows of the edge matrix?
...	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 three rules: in the "cladewise" order each clade is formed by a series of contiguous rows. In the "postorder" order, the rows are arranged so that computations following pruning-like algorithm the tree (or postorder tree traversal) can be done by descending along these rows (conversely, a preorder tree traversal can be performed by moving from the last to the first row). The "pruningwise" order is an alternative "pruning" order which is actually a bottom-up traversal order (Valiente 2002). (This third choice might be removed in the future as it merely duplicates the second one which is more efficient.) The possible multichotomies and branch lengths are preserved.

Note that for a given order, there are several possible orderings of the rows of 'edge'.

Value

an object of class "phylo" (with the attribute "order" set accordingly), or a numeric vector if index.only = TRUE; if x is of class "multiPhylo", then an object of the same class.

Author(s)

Emmanuel Paradis

References

Valiente, G. (2002) *Algorithms on Trees and Graphs.* New York: Springer.

See Also

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

Examples

```
data(bird.families)
tr <- reorder(bird.families, "postorder")
all.equal(bird.families, tr) # uses all.equal.phylo actually
all.equal.list(bird.families, tr) # bypasses the generic

## get the number of descendants for each tip or node:
nr_desc <-  function(x) {
    res <- numeric(max(x$edge))
    res[1:Ntip(x)] <- 1L
    for (i in postorder(x)) {
       tmp <- x$edge[i,1]
       res[tmp] <- res[tmp] + res[x$edge[i, 2]]
   }
   res
}
## apply it to a random tree:
tree <- rtree(10)
plot(tree, show.tip.label = FALSE)
tiplabels()
nodelabels()
nr_desc(tree)
```

---

richness.yule.test *Test of Diversification-Shift With the Yule Process*

Description

This function performs a test of shift in diversification rate using probabilities from the Yule process.

Usage

```
richness.yule.test(x, t)
```

Arguments

x	a matrix or a data frame with at least two columns: the first one gives the number of species in clades with a trait supposed to increase or decrease diversification rate, and the second one the number of species in the sister-clades without the trait. Each row represents a pair of sister-clades.
t	a numeric vector giving the divergence times of each pair of clades in x.

Value

a data frame with the χ^2, the number of degrees of freedom (= 1), and the *P*-value.

Author(s)

Emmanuel Paradis

References

Paradis, E. (2012) Shift in diversification in sister-clade comparisons: a more powerful test. *Evolution*, **66**, 288–295.

See Also

slowinskiguyer.test, mcconwaysims.test, diversity.contrast.test

Examples

```
### see example(mcconwaysims.test)
```

---

rlineage *Tree Simulation Under the Time-Dependent Birth–Death Models*

Description

These three functions simulate phylogenies under any time-dependent birth–death model: `rlineage` generates a complete tree including the species going extinct before present; `rbdtree` generates a tree with only the species living at present (thus the tree is ultrametric); `rphylo` generates a tree with a fixed number of species at present time. `drop.fossil` is a utility function to remove the extinct species.

Usage

```
rlineage(birth, death, Tmax = 50, BIRTH = NULL,
         DEATH = NULL, eps = 1e-6)
rbdtree(birth, death, Tmax = 50, BIRTH = NULL,
        DEATH = NULL, eps = 1e-6)
rphylo(n, birth, death, BIRTH = NULL, DEATH = NULL,
       T0 = 50, fossils = FALSE, eps = 1e-06)
drop.fossil(phy, tol = 1e-8)
```

Arguments

`birth`, `death`	a numeric value or a (vectorized) function specifying how speciation and extinction rates vary through time.
`Tmax`	a numeric value giving the length of the simulation.
`BIRTH`, `DEATH`	a (vectorized) function which is the primitive of `birth` or `death`. This can be used to speed-up the computation. By default, a numerical integration is done.
`eps`	a numeric value giving the time resolution of the simulation; this may be increased (e.g., 0.001) to shorten computation times.
`n`	the number of species living at present time.
`T0`	the time at present (for the backward-in-time algorithm).
`fossils`	a logical value specifying whether to output the lineages going extinct.
`phy`	an object of class `"phylo"`.
`tol`	a numeric value giving the tolerance to consider a species as extinct.

Details

These three functions use continuous-time algorithms: `rlineage` and `rbdtree` use the forward-in-time algorithms described in Paradis (2011), whereas `rphylo` uses a backward-in-time algorithm from Stadler (2011). The models are time-dependent birth–death models as described in Kendall (1948). Speciation (birth) and extinction (death) rates may be constant or vary through time according to an R function specified by the user. In the latter case, `BIRTH` and/or `DEATH` may be used if the primitives of `birth` and `death` are known. In these functions time is the formal argument and must be named `t`.

Note that `rphylo` simulates trees in a way similar to what the package **TreeSim** does, the difference is in the parameterization of the time-dependent models which is here the same than used in the two other functions. In this parameterization scheme, time is measured from past to present (see details in Paradis 2015 which includes a comparison of these algorithms).

The difference between `rphylo` and `rphylo(... fossils = TRUE)` is the same than between `rbdtree` and `rlineage`.

Value

An object of class `"phylo"`.

Author(s)

Emmanuel Paradis

References

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

Paradis, E. (2011) Time-dependent speciation and extinction from phylogenies: a least squares approach. *Evolution*, **65**, 661–672.

Paradis, E. (2015) Random phylogenies and the distribution of branching times. *Journal of Theoretical Biology*, **387**, 39–45.

Stadler, T. (2011) Simulating trees with a fixed number of extant species. *Systematic Biology*, **60**, 676–684.

See Also

`yule`, `yule.time`, `birthdeath`, `rtree`, `stree`

Examples

```
set.seed(10)
plot(rlineage(0.1, 0)) # Yule process with lambda = 0.1
plot(rlineage(0.1, 0.05)) # simple birth-death process
b <- function(t) 1/(1 + exp(0.2*t - 1)) # logistic
layout(matrix(0:3, 2, byrow = TRUE))
curve(b, 0, 50, xlab = "Time", ylab = "")
mu <- 0.07
segments(0, mu, 50, mu, lty = 2)
```

```
legend("topright", c(expression(lambda), expression(mu)),
       lty = 1:2, bty = "n")
plot(rlineage(b, mu), show.tip.label = FALSE)
title("Simulated with 'rlineage'")
plot(rbdtree(b, mu), show.tip.label = FALSE)
title("Simulated with 'rbdtree'")
```

---

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, ...)
## S3 method for class 'phylo'
root(phy, outgroup, node = NULL, resolve.root = FALSE,
     interactive = FALSE, edgelabel = FALSE, ...)
## S3 method for class 'multiPhylo'
root(phy, outgroup, ...)

unroot(phy, ...)
## S3 method for class 'phylo'
unroot(phy, collapse.singles = FALSE,
        keep.root.edge = FALSE, ...)
## S3 method for class 'multiPhylo'
unroot(phy, collapse.singles = FALSE,
        keep.root.edge = FALSE, ...)

is.rooted(phy)
## S3 method for class 'phylo'
is.rooted(phy)
## S3 method for class 'multiPhylo'
is.rooted(phy)
```

Arguments

phy	an object of class "phylo" or "multiPhylo".
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.

`interactive` if TRUE the user is asked to select the node by clicking on the tree which must be plotted.

`edgelabel` a logical value specifying whether to treat node labels as edge labels and thus eventually switching them so that they are associated with the correct edges when using `drawSupportOnEdges` (see Czech et al. 2016).

`collapse.singles` a logical value specifying wether to call `collapse.singles` before proceeding to unrooting the tree.

`keep.root.edge` a logical value. If TRUE, the `root.edge` element of the tree is added in the edge matrix as a terminal edge. The default is to delete this element.

`...` arguments passed among methods (e.g., when rooting lists of trees).

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) *of the ingroup* is used as the new root. Note that the tree is 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"` or `"multiPhylo"` for `root` and `unroot`; a logical vector for `is.rooted`.

Note

The use of `resolve.root = TRUE` together with `node =` gives an error if the specified node is the current root of the tree. This is because there is an ambiguity when resolving a node in an unrooted tree with no explicit outgroup. If the node is not the current root, the ambiguity is solved arbitrarily by considering the clade on the right of `node` (when the tree is plotted by default) as the ingroup. See a detailed explanation there:

https://www.mail-archive.com/r-sig-phylo@r-project.org/msg03805.html.

Author(s)

Emmanuel Paradis

References

Czech, L., Huerta-Cepas, J. and Stamatakis, A. (2017) A critical review on the use of support values in tree viewers and bioinformatics toolkits. *Molecular Biology and Evolution*, **34**, 1535–1542. doi:10.1093/molbev/msx055

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)

x <- setNames(rmtree(10, 10), LETTERS[1:10])
is.rooted(x)
```

---

rotate *Swapping 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.

`rotateConstr` rotates internal branches giving a constraint on the order of the tips.

Usage

```
rotate(phy, node, polytom = c(1, 2))
rotateConstr(phy, constraint)
```

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.
constraint	a vector of mode character specifying the order of the tips as they should appear when plotting the tree (from bottom to top).

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>, Emmanuel Paradis

See Also

plot.phylo, nodelabels, root, drop.tip

Examples

```
# create a random tree:
tre <- rtree(25)

# visualize labels of internal nodes:
plot(tre, use.edge.length=FALSE)
nodelabels()

# rotate clades around node 30:
tre.new <- rotate(tre, 30)

# compare the results:
par(mfrow=c(1,2)) # split graphical device
plot(tre) # plot old tre
plot(tre.new) # plot new tree
```

```
# visualize labels of terminal nodes:
plot(tre)
tiplabels()

# rotate clades containing nodes 12 and 20:
tre.new <- rotate(tre, c(12, 21))

# compare the results:
par(mfrow=c(1,2)) # split 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:
par(mfrow=c(1,2)) # devide graphical device
plot(tre) # plot old tre
plot(tre.new) # plot new tree

# a simple example for rotateConstr:
A <- read.tree(text = "((A,B),(C,D));")
B <- read.tree(text = "(((D,C),B),A);")
B <- rotateConstr(B, A$tip.label)
plot(A); plot(B, d = "l")

# something more interesting (from ?cophyloplot):
tr1 <- rtree(40)
## drop 20 randomly chosen tips:
tr2 <- drop.tip(tr1, sample(tr1$tip.label, size = 20))
## rotate the root and reorder the whole:
tr2 <- rotate(tr2, 21)
tr2 <- read.tree(text = write.tree(tr2))
X <- cbind(tr2$tip.label, tr2$tip.label) # association matrix
cophyloplot(tr1, tr2, assoc = X, space = 28)
## before reordering tr2 we have to find the constraint:
co <- tr2$tip.label[order(match(tr2$tip.label, tr1$tip.label))]
newtr2 <- rotateConstr(tr2, co)
cophyloplot(tr1, newtr2, assoc = X, space = 28)
```

`rTraitCont` *Continuous Character Simulation*

Description

This function simulates the evolution of a continuous character along a phylogeny. The calculation is done recursively from the root. See Paradis (2012, pp. 232 and 324) for an introduction.

Usage

```
rTraitCont(phy, model = "BM", sigma = 0.1, alpha = 1, theta = 0,
           ancestor = FALSE, root.value = 0, ...)
```

Arguments

phy	an object of class "phylo".
model	a character (either "BM" or "OU") or a function specifying the model (see details).
sigma	a numeric vector giving the standard-deviation of the random component for each branch (can be a single value).
alpha	if model = "OU", a numeric vector giving the strength of the selective constraint for each branch (can be a single value).
theta	if model = "OU", a numeric vector giving the optimum for each branch (can be a single value).
ancestor	a logical value specifying whether to return the values at the nodes as well (by default, only the values at the tips are returned).
root.value	a numeric giving the value at the root.
...	further arguments passed to model if it is a function.

Details

There are three possibilities to specify model:

- "BM": a Browian motion model is used. If the arguments sigma has more than one value, its length must be equal to the the branches of the tree. This allows to specify a model with variable rates of evolution. You must be careful that branch numbering is done with the tree in "postorder" order: to see the order of the branches you can use: tr <- reorder(tr, "po"); plor(tr); edgelabels(). The arguments alpha and theta are ignored.
- "OU": an Ornstein-Uhlenbeck model is used. The above indexing rule is used for the three parameters sigma, alpha, and theta. This may be interesting for the last one to model varying phenotypic optima. The exact updating formula from Gillespie (1996) are used which are reduced to BM formula if alpha = 0.
- A function: it must be of the form foo(x, l) where x is the trait of the ancestor and l is the branch length. It must return the value of the descendant. The arguments sigma, alpha, and theta are ignored.

Value

A numeric vector with names taken from the tip labels of phy. If ancestor = TRUE, the node labels are used if present, otherwise, "Node1", "Node2", etc.

Author(s)

Emmanuel Paradis

References

Gillespie, D. T. (1996) Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. *Physical Review E*, **54**, 2084–2091.

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

See Also

rTraitDisc, rTraitMult, ace

Examples

```
data(bird.orders)
rTraitCont(bird.orders) # BM with sigma = 0.1
### OU model with two optima:
tr <- reorder(bird.orders, "postorder")
plot(tr)
edgelabels()
theta <- rep(0, Nedge(tr))
theta[c(1:4, 15:16, 23:24)] <- 2
## sensitive to 'alpha' and 'sigma':
rTraitCont(tr, "OU", theta = theta, alpha=.1, sigma=.01)
### an imaginary model with stasis 0.5 time unit after a node, then
### BM evolution with sigma = 0.1:
foo <- function(x, l) {
    if (l <= 0.5) return(x)
    x + (l - 0.5)*rnorm(1, 0, 0.1)
}
tr <- rcoal(20, br = runif)
rTraitCont(tr, foo, ancestor = TRUE)
### a cumulative Poisson process:
bar <- function(x, l) x + rpois(1, l)
(x <- rTraitCont(tr, bar, ancestor = TRUE))
plot(tr, show.tip.label = FALSE)
Y <- x[1:20]
A <- x[-(1:20)]
nodelabels(A)
tiplabels(Y)
```

---

rTraitDisc *Discrete Character Simulation*

Description

This function simulates the evolution of a discrete character along a phylogeny. If `model` is a character or a matrix, evolution is simulated with a Markovian model; the transition probabilities are calculated for each branch with $P = e^{Qt}$ where Q is the rate matrix given by `model` and t is the branch length. The calculation is done recursively from the root. See Paradis (2006, p. 101) for a general introduction applied to evolution.

Usage

```
rTraitDisc(phy, model = "ER", k = if (is.matrix(model)) ncol(model) else 2,
           rate = 0.1, states = LETTERS[1:k], freq = rep(1/k, k),
           ancestor = FALSE, root.value = 1, ...)
```

Arguments

phy	an object of class "phylo".
model	a character, a square numeric matrix, or a function specifying the model (see details).
k	the number of states of the character.
rate	the rate of change used if model is a character; it is *not* recycled if model = "ARD" of model = "SYM".
states	the labels used for the states; by default "A", "B", ...
freq	a numeric vector giving the equilibrium relative frequencies of each state; by default the frequencies are equal.
ancestor	a logical value specifying whether to return the values at the nodes as well (by default, only the values at the tips are returned).
root.value	an integer giving the value at the root (by default, it's the first state). To have a random value, use root.value = sample(k).
...	further arguments passed to model if it is a function.

Details

There are three possibilities to specify model:

- A matrix: it must be a numeric square matrix; the diagonal is always ignored. The arguments k and rate are ignored.
- A character: these are the same short-cuts than in the function ace: "ER" is an equal-rates model, "ARD" is an all-rates-different model, and "SYM" is a symmetrical model. Note that the argument rate must be of the appropriate length, i.e., 1, $k(k-1)$, or $k(k-1)/2$ for the three models, respectively. The rate matrix Q is then filled column-wise.
- A function: it must be of the form foo(x, l) where x is the trait of the ancestor and l is the branch length. It must return the value of the descendant as an integer.

Value

A factor with names taken from the tip labels of phy. If ancestor = TRUE, the node labels are used if present, otherwise, "Node1", "Node2", etc.

Author(s)

Emmanuel Paradis

References

Paradis, E. (2006) *Analyses of Phylogenetics and Evolution with R.* New York: Springer.

See Also

rTraitCont, rTraitMult, ace

Examples

```
data(bird.orders)
### the two followings are the same:
rTraitDisc(bird.orders)
rTraitDisc(bird.orders, model = matrix(c(0, 0.1, 0.1, 0), 2))

### two-state model with irreversibility:
rTraitDisc(bird.orders, model = matrix(c(0, 0, 0.1, 0), 2))

### simple two-state model:
tr <- rcoal(n <- 40, br = runif)
x <- rTraitDisc(tr, ancestor = TRUE)
plot(tr, show.tip.label = FALSE)
nodelabels(pch = 19, col = x[-(1:n)])
tiplabels(pch = 19, col = x[1:n])

### an imaginary model with stasis 0.5 time unit after a node, then
### random evolution:
foo <- function(x, l) {
    if (l < 0.5) return(x)
    sample(2, size = 1)
}
tr <- rcoal(20, br = runif)
x <- rTraitDisc(tr, foo, ancestor = TRUE)
plot(tr, show.tip.label = FALSE)
co <- c("blue", "yellow")
cot <- c("white", "black")
Y <- x[1:20]
A <- x[-(1:20)]
nodelabels(A, bg = co[A], col = cot[A])
tiplabels(Y, bg = co[Y], col = cot[Y])
```

---

rTraitMult *Multivariate Character Simulation*

Description

This function simulates the evolution of a multivariate set of traits along a phylogeny. The calculation is done recursively from the root.

Usage

```
rTraitMult(phy, model, p = 1, root.value = rep(0, p), ancestor = FALSE,
           asFactor = NULL, trait.labels = paste("x", 1:p, sep = ""), ...)
```

Arguments

phy	an object of class "phylo".
model	a function specifying the model (see details).

p	an integer giving the number of traits.
root.value	a numeric vector giving the values at the root.
ancestor	a logical value specifying whether to return the values at the nodes as well (by default, only the values at the tips are returned).
asFactor	the indices of the traits that are returned as factors (discrete traits).
trait.labels	a vector of mode character giving the names of the traits.
...	further arguments passed to model if it is a function.

Details

The model is specified with an R function of the form foo(x, l) where x is a vector of the traits of the ancestor and l is the branch length. Other arguments may be added. The function must return a vector of length p.

Value

A data frame with p columns whose names are given by trait.labels and row names taken from the labels of the tree.

Author(s)

Emmanuel Paradis

See Also

rTraitCont, rTraitDisc, ace

Examples

```
## correlated evolution of 2 continuous traits:
mod <- function(x, l) {
    y1 <- rnorm(1, x[1] + 0.5*x[2], 0.1)
    y2 <- rnorm(1, 0.5*x[1] + x[2], 0.1)
    c(y1, y2)
}
set.seed(11)
tr <- makeNodeLabel(rcoal(20))
x <- rTraitMult(tr, mod, 2, ancestor = TRUE)
op <- par(mfcol = c(2, 1))
plot(x, type = "n")
text(x, labels = rownames(x), cex = 0.7)
oq <- par(mar = c(0, 1, 0, 1), xpd = TRUE)
plot(tr, font = 1, cex = 0.7)
nodelabels(tr$node.label, cex = 0.7, adj = 1)
par(c(op, oq))
```

rtree *Generate Random Trees*

Description

These functions generate trees by splitting randomly the edges (rtree and rtopology) or randomly clustering the tips (rcoal). rtree and rtopology generate general trees, and rcoal generates coalescent trees. The algorithms are described in Paradis (2012) and in a vignette in this package.

Usage

```
rtree(n, rooted = TRUE, tip.label = NULL, br = runif, equiprob = FALSE, ...)
rtopology(n, rooted = FALSE, tip.label = NULL, br = runif, ...)
rcoal(n, tip.label = NULL, br = "coalescent", ...)
rmtree(N, n, rooted = TRUE, tip.label = NULL, br = runif,
       equiprob = FALSE, ...)
rmtopology(N, n, rooted = FALSE, 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	one of the following: (i) an R function used to generate the branch lengths (rtree; use NULL to simulate only a topology), or the coalescence times (rcoal); (ii) a character to simulate a genuine coalescent tree for rcoal (the default); or (iii) a numeric vector for the branch lengths or the coalescence times.
equiprob	(new since **ape** 5.4-1) a logical specifying whether topologies are generated in equal frequencies. If, FALSE, the unbalanced topologies are generated in higher proportions than the balanced ones.
...	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 option equiprob = TRUE generates *unlabelled* topologies in equal frequencies. This is more complicated for the labelled topologies (see the vignette "RandomTopologies").

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 or rmtopology.

Author(s)

Emmanuel Paradis

References

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

See Also

stree, rlineage, vignette "RandomTopologies".

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(1)
par(mar = c(5, 4, 4, 2))
```

---

rtt *Root a Tree by Root-to-Tip Regression*

Description

This function roots a phylogenetic tree with dated tips in the location most compatible with the assumption of a strict molecular clock.

Usage

```
rtt(t, tip.dates, ncpu = 1, objective = correlation,
    opt.tol = .Machine$double.eps^0.25)
```

Arguments

t	an object of class "phylo".
tip.dates	a vector of sampling times associated to the tips of t, in the same order as t$tip.label.
ncpu	number of cores to use.
objective	one of "correlation", "rms", or "rsquared".
opt.tol	tolerance for optimization precision.

Details

This function duplicates one part the functionality of the program Path-O-Gen (see references). The root position is chosen to produce the best linear regression of root-to-tip distances against sampling times.

`t` must have branch lengths in units of expected substitutions per site.

`tip.dates` should be a vector of sampling times, in any time unit, with time increasing toward the present. For example, this may be in units of "days since study start" or "years since 10,000 BCE", but not "millions of yearsago".

Setting `ncpu` to a value larger than 1 requires the `parallel` library.

`objective` is the measure which will be used to define the "goodness" of a regression fit. It may be one of `"correlation"` (strongest correlation between tip date and distance from root), `"rms"` (lowest root-mean-squared error), or `"rsquared"` (highest R-squared value).

`opt.tol` is used to optimize the location of the root along the best branch. By default, R's `optimize` function uses a precision of `.Machine$double.eps^0.25`, which is about 0.0001 on a 64-bit system. This should be set to a smaller value if the branch lengths of `t` are very short.

Value

an object of class `"phylo"`.

Note

This function only chooses the best root. It does not rescale the branch lengths to time, or perform a statistical test of the molecular clock hypothesis.

Author(s)

Rosemary McCloskey<rmccloskey@cfenet.ubc.ca>, Emmanuel Paradis

References

Rambaut, A. (2009). Path-O-Gen: temporal signal investigation tool.

Rambaut, A. (2000). Estimating the rate of molecular evolution: incorporating non-contemporaneous sequences into maximum likelihood phylogenies. *Bioinformatics*, **16**, 395-399.

Examples

```
t <- rtree(100)
tip.date <- rnorm(t$tip.label)^2
rtt(t, tip.date)
```

SDM *Construction of Consensus Distance Matrix With SDM*

Description

This function implements the SDM method of Criscuolo et al. (2006) for a set of n distance matrices.

Usage

```
SDM(...)
```

Arguments

...	2n elements (with n > 1), the first n elements are the distance matrices: these can be (symmetric) matrices, objects of class `"dist"`, or a mix of both. The next n elements are the sequence length from which the matrices have been estimated (can be seen as a degree of confidence in matrices).

Details

Reconstructs a consensus distance matrix from a set of input distance matrices on overlapping sets of taxa. Potentially missing values in the supermatrix are represented by `NA`. An error is returned if the input distance matrices can not resolve to a consensus matrix.

Value

a 2-element list containing a distance matrix labelled by the union of the set of taxa of the input distance matrices, and a variance matrix associated to the returned distance matrix.

Author(s)

Andrei Popescu

References

Criscuolo, A., Berry, V., Douzery, E. J. P. , and Gascuel, O. (2006) SDM: A fast distance-based approach for (super)tree building in phylogenomics. *Systematic Biology*, **55**, 740–755.

See Also

`bionj`, `fastme`, `njs`, `mvrs`, `triangMtd`

`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, strict = FALSE, trailingGapsAsN = TRUE)
```

Arguments

`x`	a matrix or a list which contains the DNA sequences.
`strict`	a logical value; if `TRUE`, ambiguities and gaps in the sequences are not interpreted in the usual way.
`trailingGapsAsN`	a logical value; if `TRUE` (the default), the leading and trailing alignment gaps are considered as unknown bases (i.e., N).

Details

If the sequences are in a list, they must all be of the same length.

If `strict = FALSE` (the default), the following rule is used to determine if a site is polymorphic or not in the presence of ambiguous bases: 'A' and 'R' are not interpreted as different, 'A' and 'Y' are interpreted as different, and 'N' and any other base (ambiguous or not) are interpreted as not different. If `strict = TRUE`, all letters are considered different.

Alignment gaps are considered different from all letters except for the leading and trailing gaps if `trailingGapsAsN = TRUE` (which is the default).

Value

A numeric (integer) vector giving the indices of the segregating sites.

Author(s)

Emmanuel Paradis

See Also

`base.freq`, `theta.s`, `nuc.div` (last two in **pegas**)

Examples

```
data(woodmouse)
y <- seg.sites(woodmouse)
y
length(y)
```

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

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

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))
```

`slowinskiguyer.test` *Slowinski-Guyer Test of Homogeneous Diversification*

Description

This function performs the Slowinski–Guyer test that a trait or variable does not increase diversification rate.

Usage

```
slowinskiguyer.test(x, detail = FALSE)
```

Arguments

`x`	a matrix or a data frame with at least two columns: the first one gives the number of species in clades with a trait supposed to increase diversification rate, and the second one the number of species in the corresponding sister-clade without the trait. Each row represents a pair of sister-clades.
`detail`	if `TRUE`, the individual P-values are appended.

Details

The Slowinski–Guyer test compares a series of sister-clades where one of the two is characterized by a trait supposed to increase diversification rate. The null hypothesis is that the trait does not affect diversification. If the trait decreased diversification rate, then the null hypothesis cannot be rejected.

The present function has mainly a historical interest. The Slowinski–Guyer test generally performs poorly: see Paradis (2012) alternatives and the functions cited below.

Value

a data frame with the χ^2, the number of degrees of freedom, and the *P*-value. If `detail = TRUE`, a list is returned with the data frame and a vector of individual *P*-values for each pair of sister-clades.

Author(s)

Emmanuel Paradis

References

Paradis, E. (2012) Shift in diversification in sister-clade comparisons: a more powerful test. *Evolution*, **66**, 288–295.

Slowinski, J. B. and Guyer, C. (1993) Testing whether certain traits have caused amplified diversification: an improved method based on a model of random speciation and extinction. *American Naturalist*, **142**, 1019–1024.

See Also

balance, mcconwaysims.test, diversity.contrast.test, richness.yule.test, rc in **geiger**, shift.test in **apTreeshape**

Examples

```
### from Table 1 in Slowinski and Guyer(1993):
viviparous <- c(98, 8, 193, 36, 7, 128, 2, 3, 23, 70)
oviparous <- c(234, 17, 100, 4, 1, 12, 6, 1, 481, 11)
x <- data.frame(viviparous, oviparous)
slowinskiguyer.test(x, TRUE) # 'P ~ 0.32' in the paper
xalt <- x
xalt[3, 2] <- 1
slowinskiguyer.test(xalt)
```

---

solveAmbiguousBases *Solve Ambiguous Bases in DNA Sequences*

Description

Replaces ambiguous bases in DNA sequences (R, Y, W, ...) by A, G, C, or T.

Usage

```
solveAmbiguousBases(x, method = "columnwise", random = TRUE)
```

Arguments

x	a matrix of class "DNAbin"; a list is accepted and is converted into a matrix.
method	the method used (no other choice than the default for the moment; see details).
random	a logical value (see details).

Details

The replacements of ambiguous bases are done columwise. First, the base frequencies are counted: if no ambiguous base is found in the column, nothing is done. By default (i.e., if random = TRUE), the replacements are done by random sampling using the frequencies of the observed compatible, non-ambiguous bases. For instance, if the ambiguous base is Y, it is replaced by either C or T using their observed frequencies as probabilities. If random = FALSE, the greatest of these frequencies is

used. If there are no compatible bases in the column, equal probabilities are used. For instance, if the ambiguous base is R, and only C and T are observed, then it is replaced by either A or G with equal probabilities.

Alignment gaps are not changed; see the function `latag2n` to change the leading and trailing gaps.

Value

a matrix of class `"DNAbin"`.

Author(s)

Emmanuel Paradis

See Also

`base.freq`, `latag2n`, `dnds`

Examples

```
X <- as.DNAbin(matrix(c("A", "G", "G", "R"), ncol = 1))
alview(solveAmbiguousBases(X)) # R replaced by either A or G
alview(solveAmbiguousBases(X, random = FALSE)) # R always replaced by G
```

---

speciesTree *Species Tree Estimation*

Description

This function calculates the species tree from a set of gene trees.

Usage

```
speciesTree(x, FUN = min)
```

Arguments

`x`	a list of trees, e.g., an object of class `"multiPhylo"`.
`FUN`	a function used to compute the divergence times of each pair of tips.

Details

For all trees in `x`, the divergence time of each pair of tips is calculated: these are then 'summarized' with `FUN` to build a new distance matrix used to calculate the species tree with a single-linkage hierarchical clustering. The default for `FUN` computes the maximum tree (maxtree) of Liu et al. (2010). Using `FUN = mean` gives the shallowest divergence tree of Maddison and Knowles (2006).

Value

an object of class `"phylo"`.

Author(s)

Emmanuel Paradis

References

Liu, L., Yu, L. and Pearl, D. K. (2010) Maximum tree: a consistent estimator of the species tree. *Journal of Mathematical Biology*, **60**, 95–106.

Maddison, W. P. and Knowles, L. L. (2006) Inferring phylogeny despite incomplete lineage sorting. *Systematic Biology*, **55**, 21–30.

Examples

```
### example in Liu et al. (2010):
tr1 <- read.tree(text = "(((B:0.05,C:0.05):0.01,D:0.06):0.04,A:0.1);")
tr2 <- read.tree(text = "(((A:0.07,C:0.07):0.02,D:0.09):0.03,B:0.12);")
TR <- c(tr1, tr2)
TSmax <- speciesTree(TR) # MAXTREE
TSsha <- speciesTree(TR, mean) # shallowest divergence

kronoviz(c(tr1, tr2, TSmax, TSsha), horiz = FALSE,
         type = "c", cex = 1.5, font = 1)
mtext(c("Gene tree 1", "Gene tree 2", "Species tree - MAXTREE"),
      at = -c(7.5, 4, 1))
mtext("Species tree - Shallowest Divergence")
layout(1)
```

---

stree *Generates Systematic Regular Trees*

Description

This function generates trees with regular shapes.

Usage

```
stree(n, type = "star", tip.label = NULL)
```

Arguments

n	an integer giving the number of tips in the tree.
type	a character string specifying the type of tree to generate; four choices are possible: "star", "balanced", "left", "right", or any unambiguous abbreviation of these.
tip.label	a character vector giving the tip labels; if not specified, the tips "t1", "t2", ..., are given.

Details

The types of trees generated are:

- "star": a star (or comb) tree with a single internal node.
- "balanced": a fully balanced dichotomous rooted tree; `n` must be a power of 2 (2, 4, 8, ...).
- "left": a fully unbalanced rooted tree where the largest clade is on the left-hand side when the tree is plotted upwards.
- "right": same than above but in the other direction.

Value

An object of class `"phylo"`.

Author(s)

Emmanuel Paradis

See Also

`compute.brlen`, `rtree`

Examples

```
layout(matrix(1:4, 2, 2))
plot(stree(100))
plot(stree(128, "balanced"))
plot(stree(100, "left"))
plot(stree(100, "right"))
```

---

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)
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))) # allow 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)
## S3 method for class 'phylo'
Ntip(phy)
```

```
## S3 method for class 'multiPhylo'
Ntip(phy)

Nnode(phy, ...)
## S3 method for class 'phylo'
Nnode(phy, internal.only = TRUE, ...)
## S3 method for class 'multiPhylo'
Nnode(phy, internal.only = TRUE, ...)

Nedge(phy)
## S3 method for class 'phylo'
Nedge(phy)
## S3 method for class 'multiPhylo'
Nedge(phy)
```

Arguments

object, phy	an object of class "phylo" or "multiPhylo".
...	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.

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

See Also

read.tree, summary for the generic R function, multiphylo, c.phylo

Examples

```
data(bird.families)
summary(bird.families)
Ntip(bird.families)
Nnode(bird.families)
Nedge(bird.families)
```

trans *Translation from DNA to Amino Acid Sequences*

Description

trans translates DNA sequences into amino acids. complement returns the (reverse) complement sequences.

Usage

```
trans(x, code = 1, codonstart = 1)
complement(x)
```

Arguments

x	an object of class "DNAbin" (vector, matrix or list).
code	an integer value giving the genetic code to be used. Currently only the genetic codes 1 to 6 are supported.
codonstart	an integer giving where to start the translation. This should be 1, 2, or 3, but larger values are accepted and have for effect to start the translation further towards the 3'-end of the sequence.

Details

With trans, if the sequence length is not a multiple of three, a warning message is printed. Alignment gaps are simply ignored (i.e., AG- returns X with no special warning or message). Base ambiguities are taken into account where relevant: for instance, GGN, GGA, GGR, etc, all return G.

See the link given in the References for details about the taxonomic coverage and alternative codons of each code.

Value

an object of class "AAbin" or "DNAbin", respectively.

Note

These functions are equivalent to translate and comp in the package **seqinr** with the difference that there is no need to convert the sequences into character strings.

Author(s)

Emmanuel Paradis

References

https://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/index.cgi?chapter=cgencodes

See Also

AAbin, checkAlignment, alview

Examples

```
data(woodmouse)
X <- trans(woodmouse) # not correct
X2 <- trans(woodmouse, 2) # using the correct code
identical(X, X2)
alview(X[1:2, 1:60]) # some 'Stop' codons (*)
alview(X2[, 1:60])
X2
```

---

treePop *Tree Popping*

Description

Method for reconstructing phylogenetic trees from an object of class splits using tree popping.

Usage

```
treePop(obj)
```

Arguments

obj	an object of class "bitsplit".

Value

an object of class "phylo" which displays all the splits in the input object.

Author(s)

Andrei Popescu

trex *Tree Explorer With Multiple Devices*

Description

This function requires a plotted tree: the user is invited to click close to a node and the corresponding subtree (or clade) is plotted on a new window.

Usage

```
trex(phy, title = TRUE, subbg = "lightyellow3",
     return.tree = FALSE, ...)
```

Arguments

phy	an object of class "phylo".
title	a logical or a character string (see details).
subbg	a character string giving the background colour for the subtree.
return.tree	a logical: if TRUE, the subtree is returned after being plotted and the operation is stopped.
...	further arguments to pass to plot.phylo.

Details

This function works with a tree (freshly) plotted on an interactive graphical device (i.e., not a file). After calling trex, the user clicks close to a node of the tree, then the clade from this node is plotted on a *new* window. The user can click as many times on the main tree: the clades are plotted successively on the *same* new window. The process is stopped by a right-click. If the user clicks too close to the tips, a message "Try again!" is printed.

Each time trex is called, the subtree is plotted on a new window without closing or deleting those possibly already plotted. They may be distinguished with the options title and/or subbg.

In all cases, the device where phy is plotted is the active window after the operation. It should *not* be closed during the whole process.

If title = TRUE, a default title is printed on the new window using the node label, or the node number if there are no node labels in the tree. If title = FALSE, no title is printed. If title is a character string, it is used for the title.

Value

an object of class "phylo" if return.tree = TRUE

Author(s)

Emmanuel Paradis

See Also

plot.phylo, identify.phylo

Examples

```
## Not run:
tr <- rcoal(1000)
plot(tr, show.tip.label = FALSE)
trex(tr) # left-click as many times as you want, then right-click
tr <- makeNodeLabel(tr)
trex(tr, subbg = "lightgreen") # id.

## generate a random colour with control on the darkness:
rRGB <- function(a, b)
    rgb(runif(1, a, b), runif(1, a, b), runif(1, a, b))

### with a random pale background:
trex(tr, subbg = rRGB(0.8, 1))
## the above can be called many times...
graphics.off() # close all graphical devices

## End(Not run)
```

---

triangMtd *Tree Reconstruction Based on the Triangles Method*

Description

Fast distance-based construction method. Should only be used when distance measures are fairly reliable.

Usage

```
triangMtd(X)
triangMtds(X)
```

Arguments

X	a distance matrix

.

Value

an object of class "phylo".

Author(s)

Andrei Popescu

References

http://archive.numdam.org/ARCHIVE/RO/RO_2001__35_2/RO_2001__35_2_283_0/RO_2001__35_2_283_0.pdf

See Also

nj, bionj, fastme, njs, mvrs, SDM

Examples

```
data(woodmouse)
tr <- triangMtd(dist.dna(woodmouse))
plot(tr)
```

---

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" with an attribute "old.index" indicating which trees of the original list are similar (the tree of smaller index is taken as reference).

Author(s)

Emmanuel Paradis

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)
```

`updateLabel` *Update Labels*

Description

This function changes labels (names or rownames) giving two vectors (`old` and `new`). It is a generic function with several methods as described below.

Usage

```
updateLabel(x, old, new, ...)
## S3 method for class 'character'
updateLabel(x, old, new, exact = TRUE, ...)
## S3 method for class 'DNAbin'
updateLabel(x, old, new, exact = TRUE, ...)
## S3 method for class 'AAbin'
updateLabel(x, old, new, exact = TRUE, ...)
## S3 method for class 'phylo'
updateLabel(x, old, new, exact = TRUE, nodes = FALSE, ...)
## S3 method for class 'evonet'
updateLabel(x, old, new, exact = TRUE, nodes = FALSE, ...)
## S3 method for class 'data.frame'
updateLabel(x, old, new, exact = TRUE, ...)
## S3 method for class 'matrix'
updateLabel(x, old, new, exact = TRUE, ...)
```

Arguments

`x`	an object where to change the labels.
`old`, `new`	two vectors of mode character (must be of the same length).
`exact`	a logical value (see details).
`nodes`	a logical value specifying whether to also update the node labels of the tree or network.
`...`	further arguments passed to and from methods.

Details

This function can be used to change some of the labels (see examples) or all of them if their ordering is not sure.

If `exact = TRUE` (the default), the values in `old` are matched exactly with the labels; otherwise (`exact = FALSE`), the values in `old` are considered as regular expressions and searched in the labels with `grep`.

Value

an object of the same class than `x`.

Author(s)

Emmanuel Paradis

See Also

`makeLabel`, `makeNodeLabel`, `mixedFontLabel`, `stripLabel`, `checkLabel`

Examples

```
## Not run:
## the tree by Nyakatura & Bininda-Emonds (2012, BMC Biology)
x <- "https://static-content.springer.com/esm/art"
y <- "3A10.1186"
z <- "2F1741-7007-10-12/MediaObjects/12915_2011_534_MOESM5_ESM.NEX"
## The commande below may not print correctly in HTML because of the
## percentage symbol; see the text or PDF help page.
url <- paste(x, y, z, sep = "
TC <- read.nexus(url)
tr <- TC$carnivoreST_bestEstimate
old <- c("Uncia_uncia", "Felis_manul", "Leopardus_jacobitus")
new <- c("Panthera_uncia", "Otocolobus_manul", "Leopardus_jacobita")
tr.updated <- updateLabel(tr, old, new)

## End(Not run)

tr <- rtree(6)
## the order of the labels are randomized by this function
old <- paste0("t", 1:6)
new <- paste0("x", 1:6)
updateLabel(tr, old, new)
tr
```

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 (2002).

Value

A named vector of class varcomp with estimated variance components.

Author(s)

Julien Dutheil <dutheil@evolbio.mpg.de>

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)
library(nlme)
m <- lme(log10(SW) ~ 1, random = ~ 1|Order/SuperFamily/Family/Genus, data=carnivora)
v <- varcomp(m, TRUE, TRUE)
plot(v)
```

varCompPhylip *Variance Components with Orthonormal Contrasts*

Description

This function calls Phylip's contrast program and returns the phylogenetic and phenotypic variance-covariance components for one or several traits. There can be several observations per species.

Usage

```
varCompPhylip(x, phy, exec = NULL)
```

Arguments

x	a numeric vector, a matrix (or data frame), or a list.
phy	an object of class "phylo".
exec	a character string giving the name of the executable contrast program (see details).

Details

The data x can be in several forms: (i) a numeric vector if there is single trait and one observation per species; (ii) a matrix or data frame if there are several traits (as columns) and a single observation of each trait for each species; (iii) a list of vectors if there is a single trait and several observations per species; (iv) a list of matrices or data frames: same than (ii) but with several traits and the rows are individuals.

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.

Phylip (version 3.68 or higher) must be accessible on your computer. If you have a Unix-like operating system, the executable name is assumed to be "phylip contrast" (as in Debian); otherwise it is set to "contrast". If this doesn't suit your system, use the option exec accordingly. If the executable is not in the path, you may need to specify it, e.g., exec = "C:/Program Files/Phylip/contrast".

Value

a list with elements varA and varE with the phylogenetic (additive) and phenotypic (environmental) variance-covariance matrices. If a single trait is analyzed, these contains its variances.

Author(s)

Emmanuel Paradis

References

Felsenstein, J. (2004) Phylip (Phylogeny Inference Package) version 3.68. Department of Genetics, University of Washington, Seattle, USA. http://evolution.genetics.washington.edu/phylip/phylip.html.

Felsenstein, J. (2008) Comparative methods with sampling error and within-species variation: Contrasts revisited and revised. *American Naturalist*, **171**, 713–725.

See Also

pic, pic.ortho, compar.lynch

Examples

```
## Not run:
tr <- rcoal(30)
### Five traits, one observation per species:
x <- replicate(5, rTraitCont(tr, sigma = 1))
varCompPhylip(x, tr) # varE is small
x <- replicate(5, rnorm(30))
varCompPhylip(x, tr) # varE is large
### Five traits, ten observations per species:
x <- replicate(30, replicate(5, rnorm(10)), simplify = FALSE)
varCompPhylip(x, tr)

## End(Not run)
```

---

vcv *Phylogenetic Variance-covariance or Correlation Matrix*

Description

This function computes the expected variances and covariances of a continuous trait assuming it evolves under a given model.

This is a generic function with methods for objects of class "phylo" and "corPhyl".

Usage

```
vcv(phy, ...)
## S3 method for class 'phylo'
vcv(phy, model = "Brownian", corr = FALSE, ...)
## S3 method for class 'corPhyl'
vcv(phy, corr = FALSE, ...)
```

Arguments

phy	an object of the correct class (see above).
model	a character giving the model used to compute the variances and covariances; only "Brownian" is available (for other models, a correlation structure may be used).
corr	a logical indicating whether the correlation matrix should be returned (TRUE); by default the variance-covariance matrix is returned (FALSE).
...	further arguments to be passed to or from other methods.

Value

a numeric matrix with the names of the tips as colnames and rownames.

Note

Do not confuse this function with vcov which computes the variance-covariance matrix among parameters of a fitted model object.

Author(s)

Emmanuel Paradis

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

corBrownian, corMartins, corGrafen, corPagel, corBlomberg, vcv2phylo

Examples

```
tr <- rtree(5)
## all are the same:
vcv(tr)
vcv(corBrownian(1, tr))
vcv(corPagel(1, tr))
```

vcv2phylo *Variance-Covariance Matrix to Tree*

Description

This function transforms a variance-covariance matrix into a phylogenetic tree.

Usage

```
vcv2phylo(mat, tolerance = 1e-7)
```

Arguments

mat	a square symmetric (positive-definite) matrix.
tolerance	the numeric tolerance used to compare the branch lengths.

Details

The function tests if the matrix is symmetric and positive-definite (i.e., all its eigenvalues positive within the specified tolerance).

Value

an object of class "phylo".

Author(s)

Simon Blomberg

See Also

vcv, corPhyl

Examples

```
tr <- rtree(10)
V <- vcv(tr) # VCV matrix assuming Brownian motion
z <- vcv2phylo(V)
identical(tr, z) # FALSE
all.equal(tr, z) # TRUE
```

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.

The 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

See Also

`Moran.I`, `correlogram.formula`

where *Find Patterns in DNA Sequences*

Description

This function finds patterns in a single or a set of DNA or AA sequences.

Usage

```
where(x, pattern)
```

Arguments

x	an object inheriting the class either "DNAbin" or "AAbin".
pattern	a character string to be searched in x.

Details

If x is a vector, the function returns a single vector giving the position(s) where the pattern was found. If x is a matrix or a list, it returns a list with the positions of the pattern for each sequence.

Patterns may be overlapping. For instance, if pattern = "tata" and the sequence starts with 'tatata', then the output will be c(1, 3).

Value

a vector of integers or a list of such vectors.

Author(s)

Emmanuel Paradis

See Also

DNAbin, image.DNAbin, AAbin

Examples

```
data(woodmouse)
where(woodmouse, "tata")
## with AA sequences:
x <- trans(woodmouse, 2)
where(x, "irk")
```

---

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 tree.

Value

a numeric vector.

Author(s)

Emmanuel Paradis

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

An object of class `"DNAbin"`.

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

These functions write in a file a list of DNA sequences in sequential, interleaved, or FASTA format. write.FASTA can write either DNA or AA sequences.

Usage

```
write.dna(x, file, format = "interleaved", append = FALSE,
          nbcol = 6, colsep = " ", colw = 10, indent = NULL,
          blocksep = 1)
write.FASTA(x, file, header = NULL, append = FALSE)
```

Arguments

x	a list or a matrix of DNA sequences, or of AA sequences for write.FASTA.
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"').
header	a vector of mode character giving the header to be written in the FASTA file before the sequences. By default, there is no header.

Details

Three formats are supported in the present function: see the help page of read.dna and the references below for a description.

If the sequences have no names, then they are given "1", "2", ... as labels in the file.

With the interleaved and sequential formats, the sequences must be all of the same length. The names of the sequences are not truncated.

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 the same effect (use `indent = "\t"` for a tabulation).

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.

`write.dna(, format = "fasta")` can be very slow if the sequences are long (> 10 kb). `write.FASTA` is much faster in this situation but the formatting is not flexible: each sequence is printed on a single line, which is OK for big files that are not intended to be open with a text editor.

Value

None (invisible 'NULL').

Note

Specifying a negative value for 'nbcol' (meaning that the nucleotides are printed on a single line) gives the same output for the interleaved and sequential formats.

The names of the sequences can be truncated with the function `makeLabel`. In particular, Clustal is limited to 30 characters, and PHYML seems limited to 99 characters.

Author(s)

Emmanuel Paradis

References

Anonymous. FASTA format. https://en.wikipedia.org/wiki/FASTA_format

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`, `makeLabel`

---

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, digits = 10)
```

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.
`digits`	a numeric giving the number of digits used for printing branch lengths. For negative numbers no branch lengths are printed.

Details

If several trees are given, they must all have the same tip labels.

If among the objects given some are not trees of class `"phylo"`, they are simply skipped and not written in the file.

See `write.tree` for details on how tip (and node) labels are checked before being printed.

Value

None (invisible 'NULL').

Author(s)

Emmanuel Paradis

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.phyloXML`

write.nexus.data *Write Character Data in NEXUS Format*

Description

This function writes in a file a list of data in the NEXUS format. The names of the vectors of the list are used as taxon names.

For the moment, only sequence data (DNA or protein) are supported.

Usage

```
write.nexus.data(x, file, format = "dna", datablock = TRUE,
                 interleaved = TRUE, charsperline = NULL,
                 gap = NULL, missing = NULL)
```

Arguments

x	a matrix or a list of data each made of a single vector of mode character where each element is a character state (e.g., "A", "C", ...) Objects of class of "DNAbin" are accepted.
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. Four choices are possible: "dna" (the default) "protein", "standard" or "continuous" or any unambiguous abbreviation of these (case insensitive).
datablock	a logical, if TRUE the data are written in a single DATA block. If FALSE, the data are 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, the data are written in sequential format. Default is TRUE.
charsperline	a numeric value 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.

Value

None (invisible 'NULL').

Author(s)

Johan Nylander <nylander@scs.fsu.edu> and Thomas Guillerme

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
data(woodmouse)
write.nexus.data(woodmouse, file= "wood.ex.nex", interleaved = TRUE, charsperline = 100)
## Write sequential DNA data in TAXA and CHARACTERS blocks
data(cynipids)
write.nexus.data(cynipids, file = "cyn.ex.nex", format = "protein",
                 datablock = FALSE, interleaved = FALSE)
unlink(c("wood.ex.nex", "cyn.ex.nex"))

## End(Not run)
```

---

write.phyloXML *Write Tree File in phyloXML Format*

Description

This function writes trees to a file of phyloXML format.

Usage

```
write.phyloXML(phy, file = "", tree.names = FALSE)
```

Arguments

phy	an object of class "phylo" or "multiPhylo".
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).
tree.names	either a logical or a vector of mode character specifying whether or which tree names should be written to the file.

Details

If several trees are given, they will be represented as multiple <phylogeny> elements. Contrary to `write.nexus`, the trees need not have the same tip labels.

When `tree.names` is `TRUE`, the tree names will be always added as <name> tags to each phylogeny element. If the `phy` object is unnamed, then the names will be automatically generated from the tree indices as "tree<index>" (e.g. tree1, tree2, ...). If `tree.names` is a character vector, the specified names will be used instead.

Branch lengths, labels, and rootedness are preserved in the phyloXML file.

Value

None (invisible `NULL`).

Author(s)

Federico Marotta

References

Han, M. V. and Zmasek, C. M. (2009) phyloXML: XML for evolutionary biology and comparative genomics. *BMC Bioinformatics*, **10**, 356.

See Also

`read.tree`, `write.tree`, `read.nexus`, `write.nexus`, `read.nexus.data`, `write.nexus.data`

`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, tree.names = FALSE)
```

Arguments

`phy`	an object of class `"phylo"` or `"multiPhylo"`.
`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 (significant) digits used for printing branch lengths (see details). For negative numbers no branch lengths are printed.
tree.names	either a logical or a vector of mode character. If TRUE then any tree names will be written prior to the tree on each line. If character, specifies the name of "phylo" objects which can be written to the file.

Details

The node labels and the root edge length, if available, are written in the file.

If tree.names == TRUE then a variant of the Newick format is written for which the name of a tree precedes the Newick format tree (parentheses are eventually deleted beforehand). The tree names are taken from the names attribute if present (they are ignored if tree.names is a character vector).

The tip labels (and the node labels if present) are checked before being printed: the leading and trailing spaces, and the leading left and trailing right parentheses are deleted; the other spaces are replaced by underscores; the commas, colons, semicolons, and the other parentheses are replaced with dashes.

The argument digits gives the number of *significant* digits (not rounding). For instance, if digits = 2 the branch length 1.234e-7 is printed as 1.23e-7 (not 0).

Value

a vector of mode character if file = "", none (invisible NULL) otherwise.

Author(s)

Emmanuel Paradis, Daniel Lawson <dan.lawson@bristol.ac.uk>, and Klaus Schliep <klaus.schliep@gmail.com>

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, write.phyloXML

yule *Fits the 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

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:

$$\ln \frac{\lambda_i}{1-\lambda_i} = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \alpha$$

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 ace.

Value

A NULL value is returned, the results are simply printed. The output includes the deviance of the null (intercept-only) model and a likelihood-ratio test of the fitted model against the null model. Note that the deviance of the null model is different from the one returned by yule because of the different parametrizations.

Author(s)

Emmanuel Paradis

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)
### another example with a tree that has a multichotomy
data(bird.families)
y <- rnorm(272) # 137 tips + 135 nodes
yule.cov(bird.families, ~ y)
```

---

yule.time *Fits the Time-Dependent Yule Model*

Description

This function fits by maximum likelihood the time-dependent Yule model. The time is measured from the past (root.time) to the present.

Usage

```
yule.time(phy, birth, BIRTH = NULL, root.time = 0, opti = "nlm", start = 0.01)
```

Arguments

phy	an object of class "phylo".
birth	a (vectorized) function specifying how the birth (speciation) probability changes through time (see details).
BIRTH	a (vectorized) function giving the primitive of birth.
root.time	a numeric value giving the time of the root node (time is measured from the past towards the present).
opti	a character string giving the function used for optimisation of the likelihood function. Three choices are possible: "nlm", "nlminb", or "optim", or any unambiguous abbreviation of these.
start	the initial values used in the optimisation.

Details

The model fitted is a straightforward extension of the Yule model with covariates (see yule.cov). Rather than having heterogeneity among lineages, the speciation probability is the same for all lineages at a given time, but can change through time.

The function birth *must* meet these two requirements: (i) the parameters to be estimated are the formal arguments; (ii) time is named t in the body of the function. However, this is the opposite for the primitive BIRTH: t is the formal argument, and the parameters are used in its body. See the examples.

It is recommended to use BIRTH if possible, and required if speciation probability is constant on some time interval. If this primitive cannot be provided, a numerical integration is done with integrate.

The standard-errors of the parameters are computed with the Hessian of the log-likelihood function.

Value

An object of class "yule" (see yule).

Author(s)

Emmanuel Paradis

References

Hubert, N., Paradis, E., Bruggemann, H. and Planes, S. (2011) Community assembly and diversification in Indo-Pacific coral reef fishes. *Ecology and Evolution*, **1**, 229–277.

See Also

branching.times, ltt.plot, birthdeath, yule, yule.cov, bd.time

Examples

```
### define two models...
birth.logis <- function(a, b) 1/(1 + exp(-a*t - b)) # logistic
birth.step <- function(l1, l2, Tcl) { # 2 rates with one break-point
    ans <- rep(l1, length(t))
    ans[t > Tcl] <- l2
    ans
}
### ... and their primitives:
BIRTH.logis <- function(t) log(exp(-a*t) + exp(b))/a + t
BIRTH.step <- function(t)
{
    out <- numeric(length(t))
    sel <- t <= Tcl
    if (any(sel)) out[sel] <- t[sel] * l1
    if (any(!sel)) out[!sel] <- Tcl * l1 + (t[!sel] - Tcl) * l2
    out
}
data(bird.families)
### fit both models:
yule.time(bird.families, birth.logis)
yule.time(bird.families, birth.logis, BIRTH.logis) # same but faster
## Not run: yule.time(bird.families, birth.step)  # fails
yule.time(bird.families, birth.step, BIRTH.step,
          opti = "nlminb", start = c(.01, .01, 100))
```

---

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

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

∗ **IO**
alview, 21
getAnnotationsGenBank, 138
read.dna, 218
read.GenBank, 223
read.gff, 224
read.nexus, 226
read.tree, 229
write.dna, 280
write.nexus, 281
write.phyloXML, 284
write.tree, 285
∗ **aplot**
add.scale.bar, 15
axisPhylo, 31
edges, 131
identify.phylo, 141
ltt.plot, 157
nodelabels, 186
phydataplot, 196
∗ **arith**
howmanytrees, 140
∗ **array**
matexpo, 166
∗ **cluster**
dist.dna, 114
∗ **datagen**
rDNAbin, 216
rlineage, 236
rTraitCont, 242
rTraitDisc, 244
rTraitMult, 246
rtree, 248
stree, 260
∗ **datasets**
bird.families, 47
bird.orders, 48
carnivora, 59
chiroptera, 63
data.nex, 107
hivtree, 139
mat3, 164
mat5M3ID, 165
mat5Mrand, 165
woodmouse, 279
∗ **dplot**
varcomp, 272
∗ **file**
read.nexus.data, 227
write.nexus.data, 283
∗ **hplot**
alex, 17
cophyloplot, 90
delta.plot, 113
evonet, 132
image.DNAbin, 143
kronoviz, 149
LTT, 155
ltt.plot, 157
plot.correlogram, 205
plot.phylo, 206
plot.phylo.extra, 212
plot.varcomp, 213
plotTreeTime, 214
read.caic, 217
skylineplot, 255
subtreeplot, 261
trex, 267
zoom, 291
∗ **htest**
boot.phylo, 50
diversity.contrast.test, 123
mcconwaysims.test, 167
richness.yule.test, 235
slowinskiguyer.test, 257
∗ **logic**
is.binary, 145
∗ **manip**

AAbin, 7
additive, 16
all.equal.DNAbin, 18
all.equal.phylo, 19
apetools, 22
as.alignment, 23
as.bitsplits, 25
as.matching, 26
as.phylo, 28
as.phylo.formula, 30
balance, 32
base.freq, 33
bind.tree, 43
boot.phylo, 50
branching.times, 53
c.phylo, 54
checkLabel, 61
checkValidPhylo, 62
clustal, 70
coalescent.intervals, 73
collapse.singles, 74
collapsed.intervals, 75
comparePhylo, 84
compute.brlen, 86
compute.brtime, 87
consensus, 88
cophenetic.phylo, 89
def, 109
degree, 111
dist.dna, 114
dist.gene, 117
dist.topo, 118
DNAbin, 124
DNAbin2indel, 127
drop.tip, 129
evonet, 132
Initialize.corPhyl, 144
is.compatible, 146
label2table, 150
ladderize, 151
latag2n, 152
makeLabel, 160
makeNodeLabel, 161
mcmc.popsize, 168
mixedFontLabel, 170
mrca, 175
multi2di, 177
multiphylo, 179
node.depth, 185
nodepath, 190
print.phylo, 215
read.nexus, 226
read.tree, 229
reorder.phylo, 233
root, 238
rotate, 240
skyline, 253
solveAmbiguousBases, 258
subtrees, 262
summary.phylo, 263
unique.multiPhylo, 269
updateLabel, 270
vcv, 274
vcv2phylo, 276
weight.taxo, 277
where, 277
which.edge, 278
write.nexus, 281
write.phyloXML, 284
write.tree, 285

∗ **math**

howmanytrees, 140

∗ **models**

ace, 10
bd.ext, 34
bd.time, 35
BIONJ, 46
birthdeath, 48
chronoMPL, 64
chronopl, 66
chronos, 68
compar.ou, 82
corBlomberg, 92
corBrownian, 93
corClasses, 94
corGrafen, 95
corMartins, 97
corPagel, 98
diversi.time, 121
FastME, 135
Initialize.corPhyl, 144
Moran.I, 172
MPR, 173
mvr, 180
nj, 181
njs, 182

phymltest, 200
reconstruct, 231
SDM, 251
speciesTree, 259
treePop, 266
triangMtd, 268
yule, 287
yule.cov, 288
yule.time, 289
∗ **model**
node.dating, 183
∗ **multivariate**
CADM.global, 55
dist.dna, 114
ewLasso, 134
lmorigin, 153
mantel.test, 163
matexpo, 166
mst, 176
parafit, 191
pcoa, 193
vcv, 274
vcv2phylo, 276
∗ **nonparametric**
CADM.global, 55
∗ **package**
ape-package, 7
∗ **regression**
binaryPGLMM, 37
compar.cheverud, 77
compar.gee, 78
compar.lynch, 81
corphylo, 100
correlogram.formula, 105
Moran.I, 172
pic, 203
pic.ortho, 204
varcomp, 272
varCompPhylip, 273
∗ **univar**
base.freq, 33
cherry, 62
del.gaps, 112
diversi.gof, 120
gammaStat, 137
seg.sites, 252
∗ **utilities**
dbd, 107
is.monophyletic, 147
is.ultrametric, 148
+.phylo *(bind.tree)*, 43
.Machine, *149*
.compressTipLabel *(c.phylo)*, 54
.uncompressTipLabel *(c.phylo)*, 54
[.AAbin *(AAbin)*, 7
[.DNAbin *(DNAbin)*, 124
[.multiPhylo *(multiphylo)*, 179
[<-.multiPhylo *(multiphylo)*, 179
[[.multiPhylo *(multiphylo)*, 179
[[<-.multiPhylo *(multiphylo)*, 179
$.multiPhylo *(multiphylo)*, 179
$<-.multiPhylo *(multiphylo)*, 179

AAbin, 7, *126*, *128*, *266*, *278*
AAsubst *(AAbin)*, 7
abbreviate, *161*
abbreviateGenus *(label2table)*, 150
ace, 10, *84*, *174*, *244*, *245*, *247*, *289*
add.scale.bar, 15, *32*, *210*
additive, 16
AIC.ace *(ace)*, 10
alex, 17, *21*, *72*, *144*
all.equal, *18*, *20*
all.equal.DNAbin, 18, *21*, *61*, *72*, *144*
all.equal.phylo, 19, *85*
alview, *10*, *17*, 21, *61*, *72*, *128*, *144*, *266*
anova, *14*, *80*
anova.ace *(ace)*, 10
aov, *153*
ape *(ape-package)*, 7
ape-package, 7
apetools, 22
arecompatible *(is.compatible)*, 146
arrows, *131*, *132*
as.AAbin *(AAbin)*, 7
as.alignment, 23
as.bitsplits, 25, *52*, *147*
as.character.AAbin *(AAbin)*, 7
as.character.DNAbin *(as.alignment)*, 23
as.DNAbin, *114*, *126*, *127*
as.DNAbin *(as.alignment)*, 23
as.evonet *(evonet)*, 132
as.hclust, *28*, *29*
as.hclust.phylo *(as.phylo)*, 28
as.igraph *(as.phylo)*, 28
as.igraph.evonet *(evonet)*, 132
as.list.AAbin *(AAbin)*, 7

as.list.DNAbin (DNAbin), 124
as.matching, 26
as.matrix.AAbin (AAbin), 7
as.matrix.DNAbin (DNAbin), 124
as.network.evonet (evonet), 132
as.network.phylo (as.phylo), 28
as.networx, *134*
as.networx.evonet (evonet), 132
as.phyDat.AAbin (AAbin), 7
as.phylo, *27*, 28, *31*
as.phylo.evonet (evonet), 132
as.phylo.formula, *29*, 30
as.phylo.matching (as.matching), 26
as.prop.part (as.bitsplits), 25
axis, *31*, *32*
axisPhylo, *15*, 31, *208*, *210*

balance, 32, *167*, *258*
base.freq, 33, *113*, *252*, *259*
bd.ext, 34, *50*, *122*, *158*, *287*, *289*
bd.time, 35, *35*, *50*, *108*, *109*, *158*, *290*
binaryPGLMM, 37
bind.tree, 43, *130*, *240*, *279*
BIONJ, 46
bionj, *136*, *181–183*, *251*, *269*
bionj (BIONJ), 46
bionjs, *46*
bionjs (njs), 182
biplot.pcoa (pcoa), 193
bird.families, 47, *48*
bird.orders, *47*, 48
birthdeath, *35*, *37*, 48, *121*, *122*, *158*, *237*, *287*, *289*, *290*
bitsplits (as.bitsplits), 25
boot.phylo, 50, *212*
branching.times, *35*, *50*, 53, *74*, *88*, *121*, *122*, *137*, *158*, *287*, *289*, *290*
bydir (apetools), 22

c.AAbin (AAbin), 7
c.DNAbin (DNAbin), 124
c.multiPhylo (c.phylo), 54
c.phylo, 54, *179*, *264*
CADM (CADM.global), 55
CADM.global, 55, *55*
CADM.post, *55*
carnivora, 59
cbind.AAbin (AAbin), 7
cbind.DNAbin, *10*
cbind.DNAbin (DNAbin), 124
checkAlignment, *18*, *21*, 60, *72*, *113*, *127*, *266*
checkLabel, 61, *151*, *161*, *162*, *171*, *271*
checkValidPhylo, 62, *111*
cherry, 62
chiroptera, 63
chol2inv, *83*
chronoMPL, 64, *68*, *70*
chronopl, *65*, 66
chronos, *67*, 68, *68*
cladewise (reorder.phylo), 233
close, *219*
clustal, *18*, *21*, 70, *144*
clustalomega (clustal), 70
coalescent.intervals, 73, *76*, *139*, *254*, *256*
coef.corBlomberg (corBlomberg), 92
coef.corBrownian (corBrownian), 93
coef.corGrafen (corGrafen), 95
coef.corMartins (corMartins), 97
coef.corPagel (corPagel), 98
collapse.singles, 74, *239*
collapsed.intervals, *74*, 75, *139*, *253*, *254*
compar.cheverud, 77
compar.gee, 78, *82*, *204*
compar.lynch, *78*, *80*, 81, *84*, *204*, *274*
compar.ou, *14*, 82, *233*
comparePhylo, *20*, 84
complement (trans), 265
compute.brlen, 86, *88*, *95*, *97*, *261*
compute.brtime, 87
connection, *219*
consensus, *52*, 88
cophenetic, *90*
cophenetic.phylo, 89, *117*, *118*, *120*
cophyloplot, 90
corBlomberg, 92, *95*, *275*
corBrownian, *14*, *84*, 93, *95*, *233*, *275*
corClasses, 94, *94*, *95*, *97*, *98*, *145*
corGrafen, 95, *95*, *275*
corMartins, *84*, *95*, 97, *275*
corMatrix.corBlomberg (corBlomberg), 92
corMatrix.corBrownian (corBrownian), 93
corMatrix.corGrafen (corGrafen), 95
corMatrix.corMartins (corMartins), 97
corMatrix.corPagel (corPagel), 98
corPagel, *95*, 98, *275*
corPhyl, *276*

corPhyl (corClasses), 94
corphylo, 100
correlogram.formula, 105, *205, 206*, *277*
countBipartitions (as.bitsplits), 25
cynipids (data.nex), 107

data.nex, 107
Date, *214*
dbd, 107
dbdTime (dbd), 107
def, 109
degree, 111
del.colgapsonly (del.gaps), 112
del.gaps, *72*, 112, *127*, *144*
del.rowgapsonly (del.gaps), 112
delta.plot, 113
dendrogram, *29*
deviance.ace (ace), 10
di2multi, *86*, *146*
di2multi (multi2di), 177
dist, *114*, *116–118*, *177*
dist.aa (AAbin), 7
dist.dna, *46*, 114, *114*, *118*, *136*, *177*, *182*, *201, 202*, *220*, *224*, *279*
dist.gene, 117, *117*, *177*
dist.ml, *10*
dist.nodes (cophenetic.phylo), 89
dist.topo, *52*, *89*, 118
diversi.gof, *35*, *50*, 120, *122*, *287*, *289*
diversi.time, *35*, *50*, 121, *121*, *287*, *289*
diversity.contrast.test, 123, *236*, *258*
DNAbin, *21*, *24*, *33*, *117*, 124, *127*, *138*, *144*, *152*, *217*, *220*, *224*, *278, 279*
DNAbin2indel, 127
dnds, 127, *259*
drawSupportOnEdges, *239*
drawSupportOnEdges (plot.phylo.extra), 212
drop.fossil (rlineage), 236
drop.tip, *44*, 129, *142*, *148*, *240, 241*, *262*, *279*, *292*
drop1, *80*
drop1.compar.gee (compar.gee), 78
dyule (dbd), 107

edgelabels, *212*
edgelabels (nodelabels), 186
edges, 131, *188*, *210*
editFileExtensions (apetools), 22
efastats (clustal), 70
estimate.dates, *215*
estimate.dates (node.dating), 183
estimate.mu (node.dating), 183
evonet, 132
ewLasso, 134
expm, *13*
extract.clade (drop.tip), 129
extract.popsize (mcmc.popsize), 168

family, *288*
fancyarrows, *133*, *197*
fancyarrows (edges), 131
FastME, 135
fastme, *46*, *181, 182*, *251*, *269*
fastme (FastME), 135
fastme.bal (FastME), 135
fastme.ols (FastME), 135
find.skyline.epsilon (skyline), 253
formula, *288*
Ftab (base.freq), 33

gammaStat, *63*, 137
GC.content (base.freq), 33
getAnnotationsGenBank, 138
getMRCA, *190*
getMRCA (mrca), 175
gls, *95*
gopher.D (parafit), 191
grep, *162*, *271*
grey, *292*
grid, *144*

has.singles (collapse.singles), 74
hclust, *29*
hivtree, 139
howmanytrees, 140
HP.links (parafit), 191

identify, *142*
identify.phylo, 141, *240*, *268*
image.AAbin, *144*
image.AAbin (AAbin), 7
image.default, *143*
image.DNAbin, *17, 18*, *21*, *61*, *72*, *113*, *126, 127*, 143, *197*, *278*
Initialize.corPhyl, 144
Initialize.corStruct, *145*
integrate, *36*, *290*

is.binary, 145, *149*, *178*
is.compatible, *26*, 146
is.monophyletic, 147
is.rooted, *88*, *146*
is.rooted *(*root*)*, 238
is.ultrametric, *54*, *146*, 148
isTRUE, *18*

keep.tip *(*drop.tip*)*, 129
kronoviz, 149, *158*, *210*

label2table, 150, *161*, *162*, *171*
labels.AAbin *(*AAbin*)*, 7
labels.DNAbin *(*DNAbin*)*, 124
ladderize, 151
LargeNumber *(*howmanytrees*)*, 140
latag2n, 152, *259*
layout, *292*
legend, *85*
letterconf *(*clustal*)*, 70
lice.D *(*parafit*)*, 191
lines, *256*
lines.popsize *(*mcmc.popsize*)*, 168
lines.skyline *(*skylineplot*)*, 255
lm, *153*
lme, *272*
lmorigin, 153, *153*
locator, *15*
logLik.ace *(*ace*)*, 10
LTT, *37*, 155, *158*
ltt.coplot *(*ltt.plot*)*, 157
ltt.lines *(*ltt.plot*)*, 157
ltt.plot, *35*, *37*, *50*, *121*, *122*, *137*, *156*, 157, *254*, *287*, *289*, *290*

make.names, *161*
make.unique, *161*
makeChronosCalib *(*chronos*)*, 68
makeLabel, *61*, *151*, 160, *162*, *171*, *271*, *281*
makeNodeLabel, *61*, *151*, 161, *161*, *171*, *271*
mantel.test, 163
mat3, 164, *165*, *166*
mat5M3ID, 165, *165*, *166*
mat5Mrand, 165, *165*
matching *(*as.matching*)*, 26
matexpo, *13*, 166
mcconwaysims.test, *124*, 167, *236*, *258*
mcmc.popsize, 168
mixedFontLabel, *61*, *151*, *161*, *162*, 170, *188*, *271*
mltt.plot *(*ltt.plot*)*, 157
Moran.I, *106*, 172, *206*, *277*
MPR, *12*, *14*, 173, *233*
mrca, *148*, 175
mst, 176
multi2di, *31*, *86*, *146*, 177
multiphylo, *55*, 179, *264*
muscle *(*clustal*)*, 70
muscle5 *(*clustal*)*, 70
mvr, *46*, 180
mvrs, *251*, *269*
mvrs *(*mvr*)*, 180

NA, *13*
Nedge *(*summary.phylo*)*, 263
Nedge.evonet *(*evonet*)*, 132
new2old.phylo *(*as.phylo*)*, 28
nexus2DNAbin *(*read.nexus.data*)*, 227
nj, *46*, *136*, 181, *183*, *269*
njs, *181*, 182, *182*, *251*, *269*
nlminb, *36*, *67*
Nnode *(*summary.phylo*)*, 263
node.dating, 183
node.depth, 185
node.height *(*node.depth*)*, 185
nodelabels, *52*, *132*, *142*, *174*, 186, *197*, *210*, *240*, *241*
nodepath, 190
Ntip *(*summary.phylo*)*, 263

old2new.phylo *(*as.phylo*)*, 28
optimize, *185*

par, *31*, *32*, *158*, *187*
parafit, 191, *191*
pcoa, 193, *193*, *194*
phydataplot, 196
phylo, *134*
phylo *(*read.tree*)*, 229
phymltest, 200
pic, *80*, *82*, *84*, 203, *205*, *274*
pic.ortho, 204, *204*, *274*
plot, *158*, *177*, *206*, *210*, *213*, *256*
plot.correlogram, *106*, 205
plot.correlogramList *(*plot.correlogram*)*, 205
plot.evonet *(*evonet*)*, 132

plot.mst (mst), 176
plot.multiPhylo (plot.phylo), 206
plot.phylo, *15*, *32*, *74*, *84, 85*, *91*, *132*, *142*, *150, 151*, *186*, *188*, *197*, 206, *212*, *214, 215*, *241*, *262*, *268*, *292*
plot.phylo.extra, 212
plot.phymltest (phymltest), 200
plot.popsize (mcmc.popsize), 168
plot.prop.part (boot.phylo), 50
plot.skyline (skylineplot), 255
plot.varcomp, 213
plotBreakLongEdges (plot.phylo.extra), 212
plotTreeTime, *185*, *212*, 214
postorder (reorder.phylo), 233
predict, *79*
predict.compar.gee (compar.gee), 78
print, *216*
print.AAbin (AAbin), 7
print.ace (ace), 10
print.binaryPGLMM (binaryPGLMM), 37
print.birthdeath (birthdeath), 48
print.bitsplits (as.bitsplits), 25
print.chronos (chronos), 68
print.compar.gee (compar.gee), 78
print.comparePhylo (comparePhylo), 84
print.corphylo (corphylo), 100
print.DNAbin (DNAbin), 124
print.evonet (evonet), 132
print.LargeNumber (howmanytrees), 140
print.lmorigin (lmorigin), 153
print.multiPhylo (print.phylo), 215
print.parafit (parafit), 191
print.phylo, 215
print.phymltest (phymltest), 200
print.prop.part (boot.phylo), 50
prop.clades (boot.phylo), 50
prop.part, *26*, *89*, *120*
prop.part (boot.phylo), 50

rainbow, *197*, *292*
rbdtree (rlineage), 236
rbind.AAbin (AAbin), 7
rbind.DNAbin (DNAbin), 124
rcoal (rtree), 248
rDNAbin, 216
read.caic, 217
read.dna, *24*, *117*, *126*, 218, *224*, *226*, *279–281*
read.evonet (evonet), 132
read.FASTA, *10*
read.FASTA (read.dna), 218
read.fastq (read.dna), 218
read.GenBank, *24*, *117*, *126*, *138*, *220*, 223, *281*
read.gff, *138*, 224
read.nexus, *64*, *218*, 226, *229*, *231*, *282*, *284–286*
read.nexus.data, 227, *227*, *282*, *284, 285*
read.table, *150*, *226*
read.tree, *31*, *47, 48*, *74*, *80*, *86*, *90*, *133*, *136*, *182*, *202*, *204*, *210*, *216*, *218*, *227*, 229, *234*, *264*, *282*, *285, 286*
reconstruct, 231
reorder, *234*
reorder.evonet (evonet), 132
reorder.multiPhylo (reorder.phylo), 233
reorder.phylo, *133*, *151*, 233
richness.yule.test, *124*, 235, *258*
ring (phydataplot), 196
rlineage, 236, *249*
rmtopology (rtree), 248
rmtree (rtree), 248
root, *44*, *130*, *174*, 238, *241*, *279*
rotate, *91*, 240
rotateConstr, *91*
rotateConstr (rotate), 240
rphylo (rlineage), 236
rtopology (rtree), 248
rTraitCont, 242, *245*, *247*
rTraitDisc, 244, *244*, *247*
rTraitMult, *244, 245*, 246
rtree, *178*, *237*, 248, *261*
rtt, *185*, 249

scan, *231*
SDM, *46*, *181*, 251, *269*
seg.sites, *10*, *33*, *113*, *127*, 252
segments, *132*
simSeq, *216*
skyline, *75, 76*, *137*, *158*, *170*, 253, *255, 256*
skylineplot, *170*, *254*, 255
slowinskiguyer.test, *124*, *167*, *236*, 257
solve, *83*
solveAmbiguousBases, *128*, 258
sort.bitsplits (as.bitsplits), 25
speciesTree, 259
str.multiPhylo (print.phylo), 215

stree, *237*, *249*, 260
stripLabel, *61*, *271*
stripLabel *(*label2table*)*, 150
subtreeplot, 261, *263*
subtrees, 262, *262*
summary, *264*
summary.phylo, *55*, *179*, *216*, 263
summary.phymltest *(*phymltest*)*, 200
summary.prop.part *(*boot.phylo*)*, 50

tcoffee *(*clustal*)*, 70
text, *15*
tiplabels *(*nodelabels*)*, 186
trans, *10*, *128*, 265
treePop, 266
trex, *17*, *210*, 267
triangMtd, *251*, 268
triangMtds, *183*
triangMtds *(*triangMtd*)*, 268

ultrametric *(*additive*)*, 16
unique, *270*
unique.multiPhylo, *20*, 269
unroot *(*root*)*, 238
updateLabel, *61*, *151*, *161*, *171*, 270

varcomp, *214*, 272
varCompPhylip, *204, 205*, 273
vcov, *275*
vcv, *95*, 274, *276*
vcv.phylo, *97*
vcv2phylo, *95*, *275*, 276

weight.taxo, *173*, 277
weight.taxo2 *(*weight.taxo*)*, 277
where, 277
which.edge, *148*, 278
wilcox.test, *123*
woodmouse, *220*, 279
write.dna, *24*, *117*, *126*, *220*, *224*, 280
write.evonet *(*evonet*)*, 132
write.FASTA *(*write.dna*)*, 280
write.nexus, *226, 227*, *229*, *231*, 281, *284–286*
write.nexus.data, *227*, *229*, *282*, 283, *285*
write.phyloXML, *282*, 284, *286*
write.tree, *136*, *182*, *202*, *227*, *231*, *282*, 285, *285*

Xplor *(*apetools*)*, 22
Xplorefiles *(*apetools*)*, 22
xyplot, *213*

yule, *35*, *50*, *121, 122*, *237*, 287, *289, 290*
yule.cov, *35*, *50*, *121, 122*, *158*, *287*, 288, *290*
yule.time, *36, 37*, *108, 109*, *237*, 289

zoom, *64*, *263*, 291