Package ‘phangorn’

February 20, 2015

Title Phylogenetic Analysis in R
Version 1.99-12
Date 2015-13-02
Author Klaus Schliep, Emmanuel Paradis
Maintainer Klaus Schliep <klaus.schliep@gmail.com>
Description Phylogenetic analysis in R: Estimation of phylogenetic
trees and networks using Maximum Likelihood, Maximum Parsimony,
distance methods and Hadamard conjugation.
Depends R (>= 3.0.0), ape (>= 3.2)
Imports quadprog, igraph (>= 0.6), Matrix, fastmatch, parallel, nnls
Suggests seqLogo, seqinr, xtable, flashClust, rgl, knitr
License GPL (>= 2)
VignetteBuilder utils, knitr
URL https://github.com/KlausVigo/phangorn
Repository CRAN
NeedsCompilation yes
Date/Publication 2015-02-14 00:46:11

R topics documented:

  phangorn-package .................................................. 2
  allTrees ......................................................... 3
  Ancestors ......................................................... 4
  ancestral.pml ................................................... 5
  as.splits ........................................................ 6
  bab .............................................................. 8
  bootstrap.pml ................................................... 9
  chloroplast ....................................................... 11
  cladePar ........................................................ 11
  consensusNet ...................................................... 12
  densiTREE ......................................................... 13
### Description

Phylogenetic analysis in R (Estimation of phylogenetic trees and networks using Maximum Likelihood, Maximum Parsimony, Distance methods & Hadamard conjugation)

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

Further information is available in two vignettes.

<table>
<thead>
<tr>
<th>Trees</th>
<th>Constructing phylogenetic trees (source, pdf)</th>
</tr>
</thead>
<tbody>
<tr>
<td>phangorn-specials</td>
<td>Advanced features (source, pdf)</td>
</tr>
<tr>
<td>Ancestral</td>
<td>Ancestral sequence reconstruction (source, pdf)</td>
</tr>
</tbody>
</table>
allTrees

The first vignette (to display type vignette('Trees')) gives an introduction in phylogenetic analysis with phangorn, and the second vignette covers more advanced feature like defining special character spaces.

Author(s)
Klaus Schliep
Maintainer: Klaus Schliep <klaus.schliep@gmail.com>

References

| allTrees | Compute all trees topologies. |

Description
allTrees computes all tree topologies for rooted or unrooted trees with up to 10 tips. allTrees returns bifurcating trees.

Usage
allTrees(n, rooted = FALSE, tip.label = NULL)

Arguments
n Number of tips (<=10).
rooted Rooted or unrooted trees (default: rooted).
tip.label Tip labels.

Value
an object of class multiPhylo.

Author(s)
Klaus Schliep <klaus.schliep@gmail.com>

Examples
trees <- allTrees(5)
par(mfrow = c(3,5))
for(i in 1:15) plot(trees[[i]])
Ancestors

tree utility function

Description

Functions for describing relationships among phylogenetic nodes.

Usage

Ancestors(x, node, type=c("all","parent"))
Children(x, node)
Siblings(x, node, include.self=FALSE)
Descendants(x, node, type=c("tips","children","all"))
mrca.phylo(x, node)

Arguments

x a tree (a phylo object).
node an integer or a vector of integers corresponding to a node ID
type specify whether to return just direct children / parents or all
include.self whether to include self in list of siblings

Details

These functions are inspired by treewalk in phylobase package, but work on the S3 phylo objects. The nodes are the indices as given in edge matrix of an phylo object. From taxon labels these indices can be easily derived matching against the tip.label argument of an phylo object. see example below. All the functions allow node to be either a scalar or vector.

Value

a vector or a list containing the indices of the nodes.

See Also

treewalk, phylo

Examples

tree = rtree(10)
plot(tree, show.tip.label = FALSE)
nodeLabels()
tiplabels()
Ancestors(tree, 1:3, "all")
Children(tree, 11)
Descendants(tree, 11, "tips")
Siblings(tree, 3)
mrca.phylo(tree, 1:3)
mrca.phylo(tree, match(c("t1","t2","t3"), tree$tip))
Ancestral character reconstruction.

Description
Marginal reconstruction of the ancestral character states.

Usage
ancestral.pml(object, type = c("ml", "bayes"))
ancestral.pars(tree, data, type = c("MPR", "ACCTRAN"), cost = NULL)
pace(tree, data, type = c("MPR", "ACCTRAN"), cost = NULL)
plotAnc(tree, data, i, col=NULL, cex.pie=par("cex"), pos="bottomright", ...)

Arguments

object an object of class pml

 wysokość tree a tree, i.e. an object of class pml
data an object of class phyDat
type method used to assign characters to internal nodes, see details.
i plots the i-th character of the data.
col a vector containing the colors for all possible states.
cex.pie a numeric defining the size of the pie graphs
pos a character string defining the position of the legend
cost A cost matrix for the transitions between two states.
... Further arguments passed to or from other methods.

Details
The argument "type" defines the criterion to assign the internal nodes. For ancestral.pml so far "ml" and (empirical) "bayes" and for ancestral.pars "MPR" and "ACCTRAN" are possible.

With parsimony reconstruction one has to keep in mind that there will be often no unique solution.

For further details see vignette("Ancestral").

Value
An object of class "phyDat", containing the ancestral states of all nodes.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>
References


See Also

`pml`, `parsimony`, `ace`, `root`

Examples

```r
example(NJ)
fit = pml(tree, Laurasiatherian)
anc.ml = ancestral.pml(fit, type = "ml")
anc.p = ancestral.pars(tree, Laurasiatherian)
## Not run:
require(seqLogo)
seqLogo( t(subset(anc.ml, 48, 1:20)[[1]]), ic.scale=FALSE)
seqLogo( t(subset(anc.p, 48, 1:20)[[1]]), ic.scale=FALSE)
## End(Not run)
plotAnc(tree, anc.ml, 1)
```

---

**as.splits**

*Splits representation of graphs and trees.*

Description

`as.splits` produces a list of splits or bipartitions.

Usage

```r
as.splits(x, ...)
## S3 method for class 'phylo'
as.splits(x, ...)
## S3 method for class 'multiPhylo'
as.splits(x, ...)
## S3 method for class 'splits'
print(x, maxp =getOption("max.print"), zero.print = ".", one.print = "|", ...)
## S3 method for class 'splits'
as.prop.part(x, ...)
compatible(obj)
allSplits(k, labels = NULL)
write.nexus.splits(obj, file="", weights=NULL)
read.nexus.splits(file)
```
addConfidences(obj, phy)
presenceAbsence(x, y)
addTrivialSplits(obj)

Arguments

x An object of class phylo or multiPhylo.
y An object of class splits.
maxp integer, default from options(maxNprint), influences how many entries of large matrices are printed at all.
zero.print character which should be printed for zeroes.
one.print character which should be printed for ones.
 Further arguments passed to or from other methods.
obj an object of class splits.
k number of taxa.
labels names of taxa.
file a file name.
weights Edge weights.
phy An object of class phylo or multiPhylo.

Value

as.split returns an object of class splits, which is mainly a list of splits and some attributes. compatible return a lower triangular matrix where an 1 indicates that two splits are incompatible.

Note

The internal representation is likely to change. read.nexus.split reads in the splits block of a nexus file. It assumes that different co-variables are tab delimited and the bipartition are separated with white-space. Comments in square brackets are ignored.

Author(s)

Klaus Schliep<klaus.schliep@gmail.com>

See Also

prop.part, lento, distanceHadamard, as.networx

Examples

(sp <- as.split(rtree(5))
write.nexus.split(sp)
**Branch and bound for finding all most parsimonious trees**

**Description**

bab finds all most parsimonious trees.

**Usage**

```
bab(data, tree = NULL, trace = 1, ...)
```

**Arguments**

- `data` an object of class phyDat.
- `tree` a phylogenetic tree an object of class phylo, otherwise a pratchet search is performed.
- `trace` defines how much information is printed during optimisation.
- `...` Further arguments passed to or from other methods

**Details**

This implementation is very slow and depending on the data may take very long time. In the worst case all \((2n-5)!!\) possible trees have to be examined. For 10 species there are already 2027025 tip-labelled unrooted trees. It only uses some basic strategies to find a lower and upper bounds similar to penny from phylip. It uses a very basic heuristic approach of MinMax Squeeze (Holland et al. 2005) to improve the lower bound. On the positive side bab is not like many other implementations restricted to binary or nucleotide data.

**Value**

bab returns all most parsimonious trees in an object of class multiPhylo.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com> based on work on Liam Revell

**References**


See Also

pratchet, dfactorial

Examples

data(yeast)
dfactorial(11)
# choose only the first two genes
gene12 <- subset(yeast, , 1:3158, site.pattern=FALSE)
trees <- bab(gene12)

Description

bootstrap.pml performs (non-parametric) bootstrap analysis and bootstrap.phyDat produces a list of bootstrapped data sets. plotBS plots a phylogenetic tree with the with the bootstrap values assigned to the (internal) edges.

Usage

bootstrap.pml(x, bs = 100, trees = TRUE, multicore=FALSE, ...)
bootstrap.phyDat(x, FUN, bs = 100, mc.cores = 1L, ...)
plotBS(tree, BStrees, type="unrooted", bs.col="black", bs.adj=NULL, ...)

Arguments

x an object of class pml or phyDat.
bs number of bootstrap samples.
trees return trees only (default) or whole pml objects.
multicore logical, if TRUE analysis is performed in parallel (see details).
mc.cores The number of cores to use during bootstrap. Only supported on UNIX-alike systems.
... further parameters used by optim.pml or plot.phylo.
FUN the function to estimate the trees.
tree The tree on which edges the bootstrap values are plotted.
BStrees a list of trees (object of class "multiPhylo").
type the type of tree to plot, so far "cladogram", "phylogram" and "unrooted" are supported.
bs.col color of bootstrap support labels.
bs.adj one or two numeric values specifying the horizontal and vertical justification of the bootstrap labels.
Details

It is possible that the bootstrap is performed in parallel, with help of the multicore package. Unfortunately the multicore package does not work under windows or with GUI interfaces ("aqua" on a mac). However it will speed up nicely from the command line ("X11").

Value

`bootstrap.pml` returns an object of class `multi.phylo` or a list where each element is an object of class `pml`. `plotBS` returns silently a tree, i.e. an object of class `multi.phylo` with the bootstrap values as node labels.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

`optim.pml`, `pml`, `plot.phylo`, `consensusNet`

Examples

```r
## Not run:
data(Laurasiatherian)
dm <- dist.logDet(Laurasiatherian)
tree <- NJ(dm)
fit = pml(tree, Laurasiatherian)
fit = optim.pml(fit, TRUE)
set.seed(123)
bs <- bootstrap.pml(fit, bs=100, optNni=TRUE)
treeBS <- plotBS(fit$tree, bs)

treeMP <- pratchet(Laurasiatherian)
treeMP <- acctran(treeMP, Laurasiatherian)
set.seed(123)
BSTrees <- bootstrap.phyDat(Laurasiatherian, pratchet, bs = 100)
treeMP <- plotBS(treeMP, BSTrees, "phylogram")
add.scale.bar()
```
**chloroplast**

```r
# export tree with bootstrap values as node labels
# write.tree(treeBS)
## End(Not run)
```

---

**Chloroplast alignment**

**Description**

Amino acid alignment of 15 genes of 19 different chloroplast.

**Usage**

data(yeast)

**Examples**

data(chloroplast)
chloroplast

---

**cladePar**

**Utility function to plot.phylo**

**Description**

cladePar can help you coloring (choosing edge width/type) of clades.

**Usage**

cladePar(tree, node, edge.color = "red", tip.color = edge.color, edge.width = 1,
edge.lty = 1, x = NULL, plot = FALSE, ...)

**Arguments**

- `tree` an object of class phylo.
- `node` the node which is the common ancestor of the clade.
- `edge.color` see plot.phylo.
- `tip.color` see plot.phylo.
- `edge.width` see plot.phylo.
- `edge.lty` see plot.phylo.
- `x` the result of a previous call to cladeInfo.
- `plot` logical, if TRUE the tree is plotted.
- `...` Further arguments passed to or from other methods.
Value

A list containing the information about the edges and tips.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

plot.phylo

Examples

tree = rtree(10)
plot(tree)
nodelabels()
x = cladePar(tree, 12)
cladePar(tree, 18, "blue", "blue", x=x, plot=TRUE)

consensusNet

Computes a consensusNetwork from a list of trees Computes a networx object from a collection of splits.

Description

Computes a consensusNetwork, i.e. an object of class networx from a list of trees, i.e. an class of class multiphylo. Computes a networx object from a collection of splits.

Usage

consensusNet(obj, prob=.3, ...)

Arguments

obj An object of class multiPhylo.
prob the proportion a split has to be present in all trees to be represented in the network.
... Further arguments passed to or from other methods.

Value

consensusNet returns an object of class networx. This is just an intermediate to plot phylogenetic networks with igraph.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>
densiTTree

References


See Also

*splitsNetwork, neighborNet, lento, distanceHadamard, plot.networx*

Examples

```r
data(Laurasiatherian)
set.seed(1)
bs <- bootstrap.phyDat(Laurasiatherian, FUN = function(x)nj(dist.hamming(x)),
                       bs=50)
class(bs) <- 'multiPhylo'
cnet = consensusNet(bs, .3)
plot(cnet, "2D")
## Not run:
library(rgl)
open3d()
plot(cnet, show.tip.label=FALSE, show.nodes=TRUE)
plot(cnet, type = "2D", show.edge.label=TRUE)

## End(Not run)
```

densiTTree

Plots a densiTree.

Description

An R function to plot trees similar to those produced by DensiTree.

Usage

densiTTree(x, type = "cladogram", alpha = 1/length(x), consensus = NULL, optim = FALSE,
          scaleX = FALSE, col = 1, width = 1, cex = 0.8, ...)

Arguments

- `x`: an object of class multiPhylo.
- `type`: a character string specifying the type of phylogeny, so far "cladogram" (default) or "phylogram" (the default) are supported.
- `alpha`: parameter for semi-transparent colors.
- `consensus`: A tree which is used to define the order of the tip labels.
- `optim`: not yet used.
- `scaleX`: scale trees to have identical heights.
densiTrie

col        edge color.
width      edge width.
cex        a numeric value giving the factor scaling of the tip labels.
...        further arguments to be passed to plot.

Details

If no consensus tree is provided densiTrie computes a rooted mrp.supertree as a backbone. This should avoid too many unnecessary crossings of edges. Trees should be rooted, other wise the output may not make sense.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

densiTree is inspired from the great DensiTree program of Remco Bouckaert.

See Also

plot.phylo, plot.networx

Examples

data(Laurasiatherian)
set.seed(1)
bs <- bootstrap.phyDat(Laurasiatherian, FUN =
    function(x) upgma(dist.hamming(x)), bs=25)
# cladogram nice to show topological differences
densiTree(bs, optim=TRUE, type="cladogram", col="blue")
densiTree(bs, optim=TRUE, type="phylogram", col="green")
## Not run:
# phylogram are nice to show different age estimates
require(PhyloOrchard)
data(BinindaEmondsEtAl2007)
BinindaEmondsEtAl2007 <- .compressTipLabel(BinindaEmondsEtAl2007)
densiTree(BinindaEmondsEtAl2007, type="phylogram", col="red")
## End(Not run)
designTree

Compute a design matrix or non-negative LS

Description

`nnls.tree` estimates the branch length using non-negative least squares given a tree and a distance matrix. `designTree` and `designSplits` compute design matrices for the estimation of edge length of (phylogenetic) trees using linear models. For larger trees a sparse design matrix can save a lot of memory.

Usage

```r
designTree(tree, method = "unrooted", sparse=FALSE, ...)  
designSplits(x, splits = "all", ...)  
nnls.tree(dm, tree, rooted=FALSE, trace=1)  
nnls.phylo(x, dm, rooted=FALSE, trace=0)  
nnls.splits(x, dm, trace = 0)  
nnls.network(x, dm)
```

Arguments

- `tree`: an object of class `phylo`
- `method`: design matrix for an "unrooted" or "rooted" ultrametric tree.
- `sparse`: return a sparse design matrix.
- `x`: number of taxa.
- `splits`: one of "all", "star".
- `dm`: a distance matrix.
- `rooted`: compute a "rooted" or "unrooted" tree.
- `trace`: defines how much information is printed during optimisation.
- `...`: further arguments, passed to other methods.

Value

`nnls.tree` return a tree, i.e. an object of class `phylo`. `designTree` and `designSplits` a matrix, possibly sparse.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

`fastme`, `distanceHadamard`, `splitsNetwork`, `upgma`
Examples

```r
example(NJ)
dm <- as.matrix(dm)
y <- dm[lower.tri(dm)]
X <- designTree(tree)
lm(y~X-1)
# avoids negative edge weights
tree2 = nnls.tree(dm, tree)
```

Description

double factorial function

Usage

```r
dfactorial(x)
ldfactorial(x)
```

Arguments

- `x` a numeric scalar or vector

Value

dfactorial(x) returns the double factorial, that is \( x = 1 \times 3 \times 5 \times \ldots \times x \) and ldfactorial(x) is the natural logarithm of it.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

factorial

Examples

dfactorial(1:10)
Description

dist.hamming and dist.logDet compute pairwise distances for an object of class phyDat. dist.ml fits distances for nucleotide and amino acid models.

Usage

dist.hamming(x, ratio = TRUE, exclude="none")
dist.logDet(x)
dist.ml(x, model="JC69", exclude="none", bf=NULL, Q=NULL, ...)

Arguments

x An object of class phyDat
ratio Compute uncorrected ('p') distance or character difference.
model One of "JC69" or one of 17 amino acid models see details.
exclude One of "none", "all", "pairwise" indicating whether to delete the sites with missing data (or ambiguous states). The default is handle missing data as in pml.
bf A vector of base frequencies.
Q A vector containing the lower triangular part of the rate matrix.
... Further arguments passed to or from other methods.

Details

So far 17 amino acid models are supported ("WAG", "JTT", "LG", "Dayhoff", "cpREV", "mtmam", "mtArt", "MiZoa", "mtREV24", "VT", "RtREV", "HIVw", "HIVb", "FLU", "Blossum62", "Dayhoff_DCMut" and "JTT_DCMut") and additional rate matrices and frequencies can be supplied.

Value

an object of class dist

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Pairwise Polymorphism P-Distances from DNA Sequences

Description

This function computes a matrix of pairwise uncorrected polymorphism p-distances. Polymorphism p-distances include intra-individual site polymorphisms (2ISPs; e.g. "R") when calculating genetic distances.

Usage

dist.p(x, cost="polymorphism", ignore.indels=TRUE)

Arguments

x a matrix containing DNA sequences; this must be of class "phyDat" (use as.phyDat to convert from DNAbin objects).
cost A cost matrix or "polymorphism" for a pre defined one.
ignore.indels a logical indicating whether gaps are treated as fifth state or not. Warning, each gap site is treated as a characters, so an an indel that spans a number of base positions would be treated as multiple character states.

Details

The polymorphism p-distances (Potts et al. in press) have been developed to analyse intra-individual variant polymorphism. For example, the widely used ribosomal internal transcribed spacer (ITS) region (e.g. Alvarez and Wendel, 2003) consists of 100's to 1000's of units within array across potentially multiple nucleolus organising regions (Bailey et al., 2003; Goeker and Grimm, 2008). This can give rise to intra-individual site polymorphisms (2ISPs) that can be detected from direct-PCR sequencing or cloning. Clone consensus sequences (see Goeker and Grimm, 2008) can be analysed with this function.

Value

an object of class dist.

See Also

For more distance methods for nucleotide data see dist.dna
Author(s)
Klaus Schliep and Alastair Potts

References


See Also
dist.dna, dist.hamming

Examples
data(Laurasiatherian)
laura = as.DNAbin(Laurasiatherian)

# Dealing with indel 2ISPs
# These can be coded using an "x" in the alignment. Note # that as.character usage in the read.dna() function.
cat("3 5",
  "No305 ATRA-",
  "No304 ATAYX",
  "No306 ATAGA",
file = "exdna.txt", sep = "\n")
(ex.dna <- read.dna("exdna.txt", format = "sequential", as.character=TRUE))
(dat= phyDat(ex.dna, "USER", levels=unique(as.vector(ex.dna)))
dist.p(dat)
Distance Hadamard produces spectra of splits from a distance matrix.

Usage

distancehadamard(dm, eps=0.001)

Arguments

dm A distance matrix.
eps Threshold value for splits.

Value
distancehadamard returns a matrix. The first column contains the distance spectra, the second one the edge-spectra. If eps is positive an object of with all splits greater eps is returned.

Author(s)
Klaus Schliep <klaus.schliep@gmail.com>, Tim White

References

See Also

hadamard, lento, plot.networx

Examples

data(yeast)
dm = dist.hamming(yeast)
dm = as.matrix(dm)
fit = distanceHadamard(dm)
lento(fit)
plot(as.networx(fit))
getClans

Clans, slices and clips

Description

Functions for clanistics to compute clans, slices, clips for unrooted trees and functions to quantify the fragmentation of trees.

Usage

getClans(tree)
getClips(tree, all=TRUE)
getSlices(tree)
getDiversity(tree, x, norm=TRUE, var.names = NULL, labels="new")
diversity(tree, X)

Arguments

tree An object of class phylo or multiPhylo (getDiversity).
all A logical, return all or just the largest clip.
x An object of class phyDat.
norm A logical, return Equitability Index (default) or Shannon Diversity.
var.names A vector of variable names.
labels see details.
X a data.frame

Details

Every split in an unrooted tree defines two complementary clans. Thus for an unrooted binary tree with n leaves there are \(2n - 3\) edges, and therefore \(4n - 6\) clans (including \(n\) trivial clans containing only one leave).

Slices are defined by a pair of splits or tripartitions, which are not clans. The number of distinguishable slices for a binary tree with \(n\) tips is \(2n^2 - 10n + 12\).

A clip is a different type of partition, defining groups of leaves that are related in terms of evolutionary distances and not only topology. Namely, clips are groups of leaves for which all pairwise path-length distances are smaller than a given threshold value (Lapointe et al. 2010). There exists different numbers of clips for different thresholds, the largest (and trivial) one being the whole tree. There is always a clip containing only the two leaves with the smallest pairwise distance.

Clans, slices and clips can be used to characterize how well a vector of categorial characters (natives/intruders) fit on a tree. We will follow the definitions of Lapointe et al.(2010). A complete clan is a clan that contains all leaves of a given state/color, but can also contain leaves of another state/color. A clan is homogeneous if it only contains leaves of one state/color.

getDiversity computes either the Shannon Diversity: \(H = - \sum_{i=1}^{k} \frac{N_i}{N} \log(\frac{N_i}{N})\), \(N = \sum_{i=1}^{k} N_i\)
or the Equitability Index: \( E = \frac{H}{\log(N)} \)

where \( N_i \) are the sizes of the \( k \) largest homogeneous clans of intruders. If the categories of the data can be separated by an edge of the tree then the E-value will be zero, and maximum equitability (E=1) is reached if all intruders are in separate clans. getDiversity computes these Intruder indices for the whole tree, complete clans and complete slices. Additionally the parsimony scores (p-scores) are reported. The p-score indicates if the leaves contain only one color (p-score=0), if the the leaves can be separated by a single split (perfect clan, p-score=1) or by a pair of splits (perfect slice, p-score=2).

So far only 2 states are supported (native, intruder), however it is also possible to recode several states into the native or intruder state using contrasts, for details see section 2 in vignette("phangorn-specials"). Furthermore unknown character states are coded as ambiguous character, which can act either as native or intruder minimizing the number of clans or changes (in parsimony analysis) needed to describe a tree for given data.

Set attribute labels to "old" for analysis as in Schliep et al. (2010) or to "new" for names which are more intuitive.

diversity returns a data.frame with the parsimony score for each tree and each levels of the variables in X. X has to be a data.frame where each column is a factor and the rownames of X correspond to the tips of the trees.

Value

getClans, getSlices and getClips return a matrix of partitions, a matrix of ones and zeros where rows correspond to a clan, slice or clip and columns to tips. A one indicates that a tip belongs to a certain partition.

getDiversity returns a list with tree object, the first is a data.frame of the equitability index or Shannon divergence and parsimony scores (p-score) for all trees and variables. The data.frame has two attributes, the first is a splits object to identify the taxa of each tree and the second is a splits object containing all partitions that perfectly fit.

Author(s)

Klaus Schliep <klaus.schliep@snv.jussieu.fr>
Francois-Joseph Lapointe <francois-joseph.lapointe@umontreal.ca>

References


See Also

parsimony, Consistency index CI, Retention index RI, phyDat
**hadamard**

**Examples**

```r
set.seed(111)
tree = rtree(10)
getClans(tree)
getClips(tree, all=TRUE)
getSlices(tree)

set.seed(123)
trees = rmmtree(10, 20)
X = matrix(sample(c("red", "blue", "violet"), 100, TRUE, c(0.5, 0.4, 0.1)), ncol=5,
          dimnames=list(paste('t',1:20, sep=""), paste('Var',1:5, sep="_"))
x = phyDat(X, type = "USER", levels = c("red", "blue"), ambiguity="violet")
plot(trees[[1]], "u", tip.color = X[trees[[1]]$tip,])  # intruders are blue

(divTab <- getDiversity(trees, x, var.names=colnames(X)))
summary(divTab)
```

---

**hadamard**  
*Hadamard Matrices and Fast Hadamard Multiplication*

**Description**

A collection of functions to perform Hadamard conjugation.

**Usage**

```r
hadamard(x)
fhm(v)
h2st(obj, eps=0.001)
h4st(obj, levels = c("a","c","g","t"))
```

**Arguments**

- `x`  
a vector of length $2^n$, where $n$ is an integer.
- `v`  
a vector of length $2^n$, where $n$ is an integer.
- `obj`  
a data.frame or character matrix, typical a sequence alignment.
- `eps`  
Threshold value for splits.
- `levels`  
levels of the sequences.

**Details**

`h2st` and `h4st` perform Hadamard conjugation for 2-state (binary, RY-coded) or 4-state (DNA/RNA) data. `write.nexus.splits` writes splits returned from `h2st` or `distanceHadamard` to a nexus file, which can be processed by Spectronet or Splitstree.

**Value**

`hadamard` returns a Hadamard matrix. `fhm` returns the fast Hadamard multiplication.
Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

distanceHadamard, lento, plot.networx

Examples

```r
H = hadamard(3)
v = 1:8
H
fhm(v)

data(yeast)
# as.character(yeast)
dat2 = dat
dat2[dat=="a" | dat=="g"] = "r"
dat2[dat=="c" | dat=="t"] = "y"
dat2 = phyDat(dat2, type="USER", levels=c("r","y"), ambiguity=NULL)
fit2 = h2st(dat2)
lento(fit2)

# write.nexus.splits(fit2, file = "test.nxs")
# read this file into Spectronet or Splitstree to show the network
## Not run:
dat4 = phyDat(dat, type="USER", levels=c("a","c", "g", "t"), ambiguity=NULL)
fit4 = h4st(dat4)

par(mfrow=c(3,1))
lento(fit4[[1]], main="Transversion")
lento(fit4[[2]], main="Transition 1")
lento(fit4[[3]], main="Transition 2")

## End(Not run)
```
Laurasiatherian data (AWCMEE)

Description
Laurasiatherian RNA sequence data

Usage
data(Laurasiatherian)

Source
Data have been taken from http://www.allanwilsoncentre.ac.nz/ and were converted to R format by <klaus.schliep@gmail.com>.

Examples
data(Laurasiatherian)
str(Laurasiatherian)

lento

Lento plot

Description
The lento plot represents support and conflict of splits/bipartitions.

Usage
lento(obj, xlim = NULL, ylim = NULL, main = "Lento plot", sub = NULL, xlab = NULL, ylab = NULL, bipart=TRUE, trivial=FALSE,...)

Arguments
obj an object of class phylo, multiPhylo or splits
xlim graphical parameter
ylim graphical parameter
main graphical parameter
sub graphical parameter
xlab graphical parameter
ylab graphical parameter
bipart plot bipartition information.
trivial logical, whether to present trivial splits (default is FALSE).
... Further arguments passed to or from other methods.
### Value

lento returns a plot.

### Author(s)

Klaus Schliep &lt;klaus.schliep@gmail.com&gt;

### References


### See Also

`as.splits`, `hadamard`

### Examples

```r
data(lyeast)
yeast.ry = acgt2ry(lyeast)
splits.h = h2st(yeast.ry)
lento(splits.h, trivial=TRUE)
```

---

### Description

`midpoint` performs midpoint rooting of a tree. `pruneTree` produces a consensus tree.

### Usage

```r
midpoint(tree)
pruneTree(tree, ..., FUN = ">=")
getRoot(tree)
```

### Arguments

- `tree` an object of class `phylo`
- `FUN` a function evaluated on the nodelabels, result must be logical.
- `...` further arguments, passed to other methods.

### Details

`pruneTree` prunes back a tree and produces a consensus tree, for trees already containing nodelabels. It assumes that nodelabels are numerical or character generated from numericals, it uses `as.numeric(as.character(tree$node.labels))` to convert them. `midpoint` so far does not transform node.labels properly.
modelTest

Value
pruneTree and midpoint a tree. getRoot returns the root node.

Author(s)
Klaus Schliep <klaus.schliep@gmail.com>

See Also
consensus, root, di2multi

Examples

tree = unroot(rtree(10))
tree$node.label = c("", round(runif(tree$Nnode-1), 3))

tree2 = midpoint(tree)
tree3 = pruneTree(tree, .5)

par(mfrow = c(3,1))
plot(tree, show.node.label=TRUE)
plot(tree2, show.node.label=TRUE)
plot(tree3, show.node.label=TRUE)

Description
Comparison of different substitution models

Usage

class(modelTest)

modelTest(object, tree= NULL, model = c("JC", "F81", "K80", "HKY", "SYM", "GTR"), G = TRUE, I = TRUE, k = 4, control = pml.control(epsilon = 1e-08, maxit = 3, trace = 1), multicore = FALSE)

Arguments

tree

object

model

G

I

k

control

multicore

A list of parameters for controlling the fitting process.
logical, whether models should estimated in parallel.
Details

modelTest estimates all the specified models for a given tree and data. When the multicore package is available, the computations are done in parallel. This is only possible without GUI interface and under linux. Only nucleotide models are tested so far.

Value

A data.frame containing the log-likelihood, AIC and BIC all tested models. The data.frame has an attributes "env" which is an environment which contains all the trees, the data and the calls to allow get the estimated models, e.g. as a starting point for further analysis (see example).

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

pml, anova

Examples

```r
## Not run:
example(NJ)
(mT <- modelTest(Laurasiatherian, tree))

# some R magic
equiv <- attr(mT, "equiv")
ls(env=equiv)
(F81 <- get("F81", equiv)) # a call
eval(F81, env=env)

data(chloroplast)
(mTA <- modelTest(chloroplast, model=c("JTT", "WAG", "LG")))

## End(Not run)
```
neighborNet

Computes a neighborNet from a distance matrix

Description

Computes a neighborNet, i.e. an object of class networx from a distance matrix.

Usage

neighborNet(x, ord = NULL)

Arguments

x       a distance matrix.
ord     a circular ordering.

Details

neighborNet is still experimental. The cyclic ordering sometimes differ from the SplitsTree implementation, the ord argument can be used to enforce a certain circular ordering.

Value

neighborNet returns an object of class networx.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

splitsNetwork, consensusNet, plot.networx, lento

Examples

data(yeast)
dm <- dist.ml(yeast)
nnet <- neighborNet(dm)
plot(nnet, "2D")
Description

This function performs the neighbor-joining tree estimation of Saitou and Nei (1987). UNJ is the unweighted version from Gascuel (1997).

Usage

nj(x)
unj(x)

Arguments

x  A distance matrix.

Value

an object of class "phylo".

Author(s)

Klaus P. Schliep <klaus.schliep@gmail.com>

References


See Also

nj, dist.dna, dist.hamming, upgma, fastme

Examples

data(Laurasiatherian)
dm <- dist.ml(Laurasiatherian)
tree <- NJ(dm)
plot(tree)
Description

nni returns a list of all trees which are one nearest neighbor interchange away. rNNI and rSPR are two methods which simulate random trees which are a specified number of rearrangement apart from the input tree. Both methods assume that the input tree is bifurcating. These methods may be useful in simulation studies.

Usage

nni(tree)
rspr(tree, moves=1, n=length(moves), k=NULL)
rNNI(tree, moves=1, n=length(moves))

Arguments

tree A phylogenetic tree, object of class phylo.
moves Number of tree rearrangements to be transformed on a tree. Can be a vector
n Number of trees to be simulated.
k If defined just SPR of distance k are performed.

Value

an object of class multiPhylo.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

Examples

tree = unroot(rtree(20))
trees1 <- nni(tree)
trees2 <- rspr(tree, 2, 10)
**parsimony**

**Parsimony tree.**

**Description**

`parsimony` returns the parsimony score of a tree using either the sankoff or the fitch algorithm. `optim.parsimony` tries to find the maximum parsimony tree using either Nearest Neighbor Interchange (NNI) rearrangements or sub tree pruning and regrafting (SPR). `pratchet` implements the parsimony ratchet (Nixon, 1999) and is the preferred way to search for the best tree. `random.addition` can be used to produce starting trees. `CI` and `RI` computes the consistency and retention index.

**Usage**

```
parsimony(tree, data, method="fitch", ...)  
optim.parsimony(tree, data, method="fitch", cost=NULL, trace=1, rearrangements="SPR", ...)  
pratchet(data, start=NULL, method="fitch", maxit=100, k=5, trace=1, all=FALSE, rearrangements="SPR", ...)  
fitch(tree, data, site = "pscore")  
sankoff(tree, data, cost = NULL, site = "pscore")  
random.addition(data, method="fitch")  
CI(tree, data, cost = NULL)  
RI(tree, data, cost = NULL)  
acctran(tree, data)
```

**Arguments**

- `data`: A object of class phyDat containing sequences.
- `tree`: tree to start the nni search from.
- `method`: one of 'fitch' or 'sankoff'.
- `cost`: A cost matrix for the transitions between two states.
- `site`: return either 'pscore' or 'site' wise parsimony scores.
- `trace`: defines how much information is printed during optimisation.
- `rearrangements`: SPR or NNI rearrangements.
- `start`: a starting tree can be supplied.
- `maxit`: maximum number of iterations in the ratchet.
- `k`: number of rounds ratchet is stopped, when there is no improvement.
- `all`: return all equally good trees or just one of them.
- `...`: Further arguments passed to or from other methods (e.g. model="sankoff" and cost matrix).

**Details**

The "SPR" rearrangements are so far only available for the "fitch" method, "sankoff" only uses "NNI". The "fitch" algorithm only works correct for binary trees.
Value

parsimony returns the maximum parsimony score (pscore). optim.parsimony returns a tree after NNI rearrangements. pratchet returns a tree or list of trees containing the best tree(s) found during the search. acctran returns a tree with edge length according to the ACCTRAN criterion.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

bab, ancestral.pml, nni, NJ.pml, getClans, ancestral.pars, bootstrap.pml

Examples

```r
set.seed(3)
data(Laurasiatherian)
dm = dist.hamming(Laurasiatherian)
tree = NJ(dm)
parsimony(tree, Laurasiatherian)
treeRA <- random.addition(Laurasiatherian)
treeNNI <- optim.parsimony(tree, Laurasiatherian)
treeRatchet <- pratchet(Laurasiatherian, start=tree)
# assign edge length
treeRatchet <- acctran(treeRatchet, Laurasiatherian)
plot(midpoint(treeRatchet))
add.scale.bar(0, 8, length=100)
parsimony(c(tree,treeNNI, treeRatchet), Laurasiatherian)
```

Description

These functions transform several DNA formats into the phyDat format. allSitePattern generates an alignment of all possible site patterns.
Usage

phyDat(data, type = "DNA", levels = NULL, return.index=TRUE, ...
read.phyDat(file, format="phylip", type="DNA", ...
write.phyDat(x, file, format="phylip",...)
## S3 method for class 'DNAbin'
as.phyDat(x, ...)
## S3 method for class 'phyDat'
as.character(x, allLevels = TRUE, ...)
## S3 method for class 'phyDat'
as.data.frame(x, ...)
## S3 method for class 'phyDat'
as.DNAbin(x, ...)
## S3 method for class 'phyDat'
snset(x, subset, select, site.pattern = TRUE, ...)
allSitePattern(n, levels=c("a","c","g","t"), names=NULL)
acgtRry(obj)
baiFreq(obj, freq=FALSE, drop.unused.levels=FALSE)

Arguments

data   An object containing sequences.
x      An object containing sequences.
type   Type of sequences ("DNA", "AA", "CODON" or "USER").
levels Level attributes.
return.index If TRUE returns a index of the site patterns.
file   A file name.
format File format of the sequence alignment (see details).
n      Number of sequences.
names  Names of sequences.
subset a subset of taxa.
select a subset of characters.
site.pattern select site pattern or sites.
allLevels return original data.
obj    as object of class phyDat
freq   logical, if 'TRUE', frequencies or counts are returned otherwise proportions
drop.unused.levels logical, drop unused levels
...    further arguments passed to or from other methods.

Details

If type "USER" a vector has to be give to levels. For example c("a", "c", "g", "t", ") would create a data object that can be used in phylogenetic analysis with gaps as fifth state. allSitePattern
returns all possible site patterns and can be useful in simulation studies. For further details see the vignette phangorn-specials.

write.phyDat calls the function write.dna or write.nexus.data and read.phyDat calls the function read.dna, read.aa or read.nexus.data see for more details over there.

You may import data directly with read.dna or read.nexus.data and convert the data to class phyDat.

The generic function c can be used to to combine sequences and unique to get all unique sequences or unique haplotypes.

acgt2ry converts a phyDat object of nucleotides into an binary ry-coded dataset.

There is a more detailed example for specifying USER defined data formats in the vignette advanced features.

Value

The functions return an object of class phyDat.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

DNabin, as.DNAbin, read.dna, read.aa and read.nexus.data and the example of pmlMix for the use of allSitePattern

Examples

data(Laurasiatherian)
class(Laurasiatherian)
Laurasiatherian
baseFreq(Laurasiatherian)
subset(Laurasiatherian, subset=1:5)
# transform into old ape format
LauraChar <- as.character(Laurasiatherian)
# and back
Laura <- phyDat(LauraChar, return.index=TRUE)
all.equal(Laurasiatherian, Laura)
allSitePattern(5)

plot.networx  Phylogenetic networks

Description

as.networx convert splits objects into a networx object. plot.networx plot phylogenetic net-
Usage

```r
as.networx(x, ...)     ## S3 method for class 'splits'
as.networx(x, planar = FALSE, ...)  ## S3 method for class 'networx'
plot(x, type="3D", use.edge.length = TRUE, show.tip.label=TRUE,
     show.edge.label=FALSE, edge.label = NULL, show.node.label=FALSE,
     node.label = NULL, show.nodes=FALSE, tip.color="blue",
     edge.color="grey", edge.width=3, edge.lty=1, font=3,
     cex=1, ...)            
```

Arguments

- **x**: an object of class "splits" (as.networx) or "networx" (plot)
- **planar**: logical whether to produce a planar graph from only cyclic splits (may excludes splits).
- **type**: "3D" to plot using rgl or "2D" in the normal device.
- **use.edge.length**: a logical indicating whether to use the edge weights of the network to draw the branches (the default) or not.
- **show.tip.label**: a logical indicating whether to show the tip labels on the graph (defaults to TRUE, i.e. the labels are shown).
- **show.edge.label**: a logical indicating whether to show the tip labels on the graph.
- **edge.label**: an additional vector of edge labels (normally not needed).
- **show.node.label**: a logical indicating whether to show the node labels (see example).
- **node.label**: an additional vector of node labels (normally not needed).
- **show.nodes**: a logical indicating whether to show the nodes (see example).
- **tip.color**: the colors used for the tip labels.
- **edge.color**: the colors used to draw edges.
- **edge.width**: the width used to draw edges.
- **edge.lty**: a vector of line types.
- **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 labels.
- **...**: Further arguments passed to or from other methods.

Details

A networx object hold the information for a phylogenetic network and extends the phylo object. Therefore some generic function for phylo objects will also work for networx objects. The argument planar = FALSE will create a planar split graph based on a cyclic ordering. These objects can be nicely plotted in "2D". So far not all parameters behave the same on the rgl "3D" and basic graphic "2D" device.
Note

The internal representation is likely to change.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References


See Also

consensusNet, neighborNet, splitsNetwork, hadamard, distanceHadamard, layout.kamada.kawai, evonet, as.igraph, densiTree

Examples

```r
set.seed(1)
tree1 = rtree(20, rooted=FALSE)
sp = as.splits(rNNI(tree1, n=10))
net = as.networx(sp)
plot(net)
## Not run:
# also see example in consensusNet
eexample(consensusNet)
## End(Not run)
```

---

**pml**

*Likelihood of a tree.*

**Description**

*pml* computes the likelihood of a phylogenetic tree given a sequence alignment and a model. *optim.pml* optimizes the different model parameters.

**Usage**

```r
pml(tree, data, bf=NULL, Q=NULL, inv=0, k=1, shape=1, rate=1, model="", ...)
optim.pml(object, optNni=FALSE, optBf=FALSE, optQ=FALSE, optInv=FALSE, optGamma=FALSE, optEdge=TRUE, optRate=FALSE, optRooted=FALSE, control = pml.control(epsilon=1e-08, maxit=10, trace=1), model = NULL, subs = NULL, ...)
pml.control(epsilon = 1e-08, maxit = 10, trace = 1)
```
Arguments

- tree: A phylogenetic tree, object of class phylo.
- data: An alignment, object of class phyDat.
- bf: Base frequencies.
- Q: A vector containing the lower triangular part of the rate matrix.
- inv: Proportion of invariable sites.
- k: Number of intervals of the discrete gamma distribution.
- shape: Shape parameter of the gamma distribution.
- rate: Rate.
- model: Allows to choose an amino acid model or nucleotide model, see details.
- object: An object of class pml.
- optNni: Logical value indicating whether topology gets optimized (NNI).
- optBf: Logical value indicating whether base frequencies get optimized.
- optQ: Logical value indicating whether rate matrix gets optimized.
- optInv: Logical value indicating whether proportion of variable size gets optimized.
- optGamma: Logical value indicating whether gamma rate parameter gets optimized.
- optEdge: Logical value indicating whether edge lengths get optimized.
- optRate: Logical value indicating whether overall rate gets optimized.
- optRooted: Logical value indicating whether edge lengths of a rooted tree get optimized.
- control: A list of parameters for controlling the fitting process.
- subs: A (integer) vector same length as Q to specify the optimization of Q.
- ...: Further arguments passed to or from other methods.
- epsilon: Stop criterion for optimization (see details).
- maxit: Maximum number of iterations (see details).
- trace: Show output during optimization (see details).

Details

The topology search uses a nearest neighbor interchange (NNI) and the implementation is similar to phyML. The option model in pml is only used for amino acid models. The option model defines the nucleotide model which is getting optimized; all models which are included in modeltest can be chosen. Setting this option (e.g. "K81" or "GTR") overrules options optBf and optQ. Here is a overview how to estimate different phylogenetic models with pml:

<table>
<thead>
<tr>
<th>Model</th>
<th>optBf</th>
<th>optQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jukes-Cantor</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>F81</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>symmetric</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>GTR</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Via model in optim.pml the following nucleotide models can be specified: JC, F81, K80, HKY, TrNe, TrN, TPM1, K81, TPM1u, TPM2, TPM2u, TPM3, TPM3u, TIM1e, TIM1, TIM2e, TIM2, TIM3e, TIM3, TVMe, TVM, SYM and GTR. These models are specified as in Posada (2008).

So far 17 amino acid models are supported ("W AG", "JTT", "LG", "Dayhoff", "cpREV", "mtmam", "mtArt", "MtZoa", "mtREV24", "VT","RtREV", "HIVw", "HIVb", "FLU", "Blossum62", "Dayhoff_DCMut" and "JTT_DCMut") and additionally rate matrices and amino acid frequencies can be supplied.

If the option 'optRooted' is set to TRUE than the edge lengths of rooted tree are optimized. The tree has to be rooted and by now ultrametric! Optimising rooted trees is generally much slower.

pmlNcontrol controls the fitting process. epsilon and maxit are only defined for the most outer loop, this affects pmlCluster, pmlPart and pmlMix. epsilon is defined as (logLik(k)-logLik(k+1))/logLik(k+1), this seems to be a good heuristic which works reasonalby for small and large trees or alignments. If trace is set to zero than no out put is shown, if functions are called internally than the trace is decreased by one, so a higher of trace produces more feedback.

**Value**

- Returns a list of class 'llNphylo'
  - logLik: Log likelihood of the tree.
  - siteLik: Site log likelihoods.
  - root: Likelihood in the root node.
  - weight: Weight of the site patterns.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**


**See Also**

bootstrap.pml, modelTest, pmlPart, pmlMix, plot.phylo, SH.test

**Examples**

```r
example(NJ)
# Jukes-Cantor (starting tree from NJ)
fitJC <- pml(tree, Laurasiatherian)
# optimize edge length parameter
fitJC <- optim.pml(fitJC)
fJC

## Not run:
# search for a better tree using NNI rearrangements
fitJC <- optim.pml(fitJC, optNni=TRUE)
fJC
plot(fitJC$tree)

# JC + Gamma + I - model
fitJC_GI <- update(fitJC, k=4, inv=.2)
# optimize shape parameter + proportion of invariant sites
fitJC_GI <- optim.pml(fitJC_GI, optGamma=TRUE, optInv=TRUE)
# GTR + Gamma + I - model
fitGTR <- optim.pml(fitJC_GI, optNni=TRUE, optGamma=TRUE, optInv=TRUE, model="GTR")

## End(Not run)

# 2-state data (RY-coded)
dat <- acgt2ry(Laurasiatherian)
fit2ST <- pml(tree, dat)
fit2ST <- optim.pml(fit2ST, optNni=TRUE)
fit2ST
# show some of the methods available for class pml
methods(class="pml")
```

**pml.fit**

*Internal maximum likelihood functions.*

**Description**

These functions are internally used for the likelihood computations in pml or optim.pml.
Usage

```
pml.fit(tree, data, bf=rep(1/length(levels), length(levels)), shape=1, k=1,
Q=rep(1, length(levels))*(length(levels)-1)/2), levels=attr(data, "levels"),
inv=0, rate=1, g=NULL, w=NULL, eig=NULL, INV=NULL, ll.0=NULL, llMix=NULL,
wMix=0, ..., site=FALSE)
pml.init(data, k)
pml.free()
edQt(Q = c(1, 1, 1, 1, 1), bf = c(0.25, 0.25, 0.25, 0.25))
lli(data, tree, ...)
```

Arguments

- `tree` A phylogenetic tree, object of class `phylo`.
- `data` An alignment, object of class `phyDat`.
- `bf` Base frequencies.
- `shape` Shape parameter of the gamma distribution.
- `k` Number of intervals of the discrete gamma distribution.
- `Q` A vector containing the lower triangular part of the rate matrix.
- `levels` Proportion of invariable sites.
- `inv` Rate.
- `rate` Rate.
- `g` Rate.
- `w` Rate.
- `eig` Eigenvalue decomposition of Q.
- `INV` Sparse representation of invariant sites.
- `ll.0` Sparse representation of invariant sites.
- `llMix` Sparse representation of invariant sites.
- `wMix` Sparse representation of invariant sites.
- `...` Further arguments passed to or from other methods.
- `site` Site

Details

These functions are exported to be used in different packages so far only in the package coalescentMCMC, but are not intended for end user. Most of the functions call C code.

Value

`pml.fit` returns the loglikelihood.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>
References


See Also

`pml`, `pmlPart`, `pmlMix`

---

**Stochastic Partitioning**

**Description**

Stochastic Partitioning of genes into p cluster.

**Usage**

```r
pmlCluster(formula, fit, weight, p=1:5, part=NULL, nrep = 10, control=pml.control(epsilon=1e-8, maxit=10, trace=1),...)
```

**Arguments**

- `formula`: a formula object (see details).
- `fit`: an object of class `pml`.
- `weight`: weight is matrix of frequency of site patterns for all genes.
- `p`: number of clusters.
- `part`: starting partition, otherwise a random partition is generated.
- `nrep`: number of replicates for each p.
- `control`: A list of parameters for controlling the fitting process.
- `...`: Further arguments passed to or from other methods.

**Details**

The formula object allows to specify which parameter get optimized. The formula is generally of the form `edge + bf + Q ~ rate + shape + ...`, on the left side are the parameters which get optimized over all cluster, on the right the parameter which are optimized specific to each cluster. The parameters available are "nni", "bf", "Q", "inv", "shape", "edge", "rate". Each parameter can be used only once in the formula. There are also some restriction on the combinations how parameters can get used. "rate" is only available for the right side. When "rate" is specified on the left hand side "edge" has to be specified (on either side), if "rate" is specified on the right hand side it follows directly that edge is too.
Value

pmlCluster returns a list with elements

- logLik: log-likelihood of the fit
- trees: a list of all trees during the optimization.
- fits: fits for the final partitions

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

pml, pmlPart, pmlMix, SH.test

Examples

```r
## Not run:
data(yeast)
dm <- dist.logDet(yeast)
tree <- NJ(dm)
fit = pml(tree, yeast)
fit = optim.pml(fit)

weight = xtabs(~ index + genes, attr(yeast, "index"))
set.seed(1)

sp <- pmlCluster(edge-rate, fit, weight, p=1:4)
sp
SH.test(sp)

## End(Not run)
```

pmlMix

**Phylogenetic mixture model**

Description

Phylogenetic mixture model.

Usage

```r
pmlMix(formula, fit, m=2, omega=rep(1/m, m), control=pml.control(epsilon=1e-08, maxit=20, trace=1),...)
```
Arguments

formula a formula object (see details).
fit an object of class pml.
m number of mixtures.
omega mixing weights.
control A list of parameters for controlling the fitting process.
... Further arguments passed to or from other methods.

Details

The formula object allows to specify which parameter get optimized. The formula is generally of the form \( edge + bf + Q \sim rate + shape + \ldots \), on the left side are the parameters which get optimized over all mixtures, on the right the parameter which are optimized specific to each mixture. The parameters available are "nni", "bf", "Q", "inv", "shape", "edge", "rate". Each parameters can be used only once in the formula. "rate" and "nni" are only available for the right side of the formula. On the other hand parameters for invariable sites are only allowed on the left-hand side. The convergence of the algorithm is very slow and is likely that the algorithm can get stuck in local optima.

Value

pmlMix returns a list with elements

logLik log-likelihood of the fit
omega mixing weights.
fits fits for the final mixtures.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

pml.pmlPart.pmlCluster

Examples

```r
## Not run:
X <- allSitePattern(5)
tree <- read.tree(text = "((t1:0.3,t2:0.3):0.1,(t3:0.3,t4:0.3):0.1,t5:0.5);")
fit <- pml(tree,X, k=4)
weights <- 1000*exp(fit$site)
attr(X, "weight") <- weights
fit1 <- update(fit, data=X, k=1)
fit2 <- update(fit, data=X)

(fitMixture <- pmlMix(edge=rate, fit1 , m=4))
(fit2 <- optim.pml(fit2, optGamma=TRUE))
```
```r
# simulation of mixture models

\dontrun{
X <- allSitePattern(5)
tree1 <- read.tree(text = "((t1:0.1,t2:0.5):0.1,(t3:0.1,t4:0.5):0.1,t5:0.5);")
tree2 <- read.tree(text = "((t1:0.5,t2:0.1):0.1,(t3:0.5,t4:0.1):0.1,t5:0.5);")
tree1 <- unroot(tree1)
tree2 <- unroot(tree2)
fit1 <- pml(tree1, X)
fit2 <- pml(tree2, X)
weights <- 2000*exp(fit1$site) + 1000*exp(fit2$site)
attr(X, "weight") <- weights
fit1 <- pml(tree1, X)
fit2 <- optim.pml(fit1)
logLik(fit2)
AIC(fit2, k=log(3000))
fitMixEdge = pmlMix( ~ edge, fit1, m=2)
logLik(fitMixEdge)
AIC(fitMixEdge, k=log(3000))
fit.p <- pmlPen(fitMixEdge, .25)
logLik(fit.p)
AIC(fit.p, k=log(3000))
}
```

---

**pmlPart**  
*Partition model.*
Description

Model to estimate phylogenies for partitioned data.

Usage

\[
pmlPart(formula, object, control = pml.control(epsilon=1e-8, maxit=10, trace=1),
model=NULL, rooted=FALSE, ...)
\]

Arguments

- **formula**: a formula object (see details).
- **object**: an object of class `pml` or a list of objects of class `pml`.
- **control**: A list of parameters for controlling the fitting process.
- **model**: A vector containing the models containing a model for each partition.
- **rooted**: Are the gene trees rooted (ultrametric) or unrooted.
- **...**: Further arguments passed to or from other methods.

Details

The formula object allows to specify which parameter get optimized. The formula is generally of the form `edge + bf + Q ~ rate + shape + ...`, on the left side are the parameters which get optimized over all partitions, on the right the parameter which are optimized specific to each partition. The parameters available are "nni", "bf", "Q", "inv", "shape", "edge", "rate". Each parameters can be used only once in the formula. "rate" and "nni" are only available for the right side of the formula.

For partitions with different edge weights, but same topology, \textit{pmlPen} can try to find more parsimonious models (see example).

Value

- \textit{kcluster} returns a list with elements
  - **logLik**: log-likelihood of the fit
  - **trees**: a list of all trees during the optimization.
  - **object**: an object of class "pml" or "pmlPart"

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

- \textit{pml, pmlCluster, pmlMix, SH.test}
Examples

data(yeast)
dm <- dist.logDet(yeast)
tree <- NJ(dm)
fit <- pml(tree, yeast)
fits <- optim.pml(fit)

weight <- xtabs(~ index+genes, attr(yeast, "index"))[,1:10]

sp <- pmlPart(edge ~ rate + inv, fits, weight=weight)
sp

## Not run:
sp2 <- pmlPart(~ edge + inv, fits, weight=weight)
sp2
AIC(sp2)

sp3 <- pmlPen(sp2, lambda = 2)
AIC(sp3)

## End(Not run)

---

**Read Amino Acid Sequences in a File**

**Description**

This function reads amino acid sequences in a file, and returns a matrix list of DNA sequences with the names of the taxa read in the file as row names.

**Usage**

```r
read.aa(file, format = "interleaved", skip = 0,
nlines = 0, comment.char = ",", seq.names = NULL)
```

**Arguments**

- `file`: a file name specified by either a variable of mode character, or a double-quoted string.
- `format`: a character string specifying the format of the DNA sequences. Three choices are possible: "interleaved", "sequential", or "fasta", or any unambiguous abbreviation of these.
- `skip`: the number of lines of the input file to skip before beginning to read data.
- `nlines`: the number of lines to be read (by default the file is read until its end).
- `comment.char`: a single character, the remaining of the line after this character is ignored.
- `seq.names`: the names to give to each sequence; by default the names read in the file are used.
**Value**

a matrix of amino acid sequences.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**


**See Also**

`read.dna`, `read.GenBank`, `phyDat`, `read.alignment`

**SH.test**

**Shimodaira-Hasegawa Test**

**Description**

This function computes the Shimodaira–Hasegawa test for a set of trees.

**Usage**

`SH.test(..., B = 10000, data=NULL)`

**Arguments**

- `...`: either a series of objects of class "pml" separated by commas, a list containing such objects or an object of class "pmlPart".
- `B`: the number of bootstrap replicates.
- `data`: an object of class "phyDat".

**Value**

a numeric vector with the P-value associated with each tree given in `...`.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**

**simSeq**

**Simulate sequences.**

**Description**

Simulate sequences for a given evolutionary tree.

**Usage**

```r
simSeq(x, ...)  # S3 method for class 'phylo'
simSeq(x, l=1000, Q=NULL, bf=NULL, rootseq=NULL, type="DNA", model="", levels=NULL, rate=1, ancestral=FALSE, ...)
```

**Arguments**

- `x` a phylogenetic tree, i.e. an object of class `phylo` or and object of class `pml`.
- `l` length of the sequence to simulate.
- `Q` the rate matrix.
- `bf` base frequencies.
- `rootseq` a vector of length l containing the root sequence, other root sequence is randomly generated.
- `type` Type of sequences ("DNA", "AA" or "USER").
model Amino acid models: one of "WAG", "JTT", "Dayhoff" or "LG"
levels levels takes a character vector of the different bases, default is for nucleotide sequences, only used when type = "USER".
rate rate.
ancestral Return ancestral sequences?
... Further arguments passed to or from other methods.

Details

simSeq is now a generic function to simulate sequence alignments. It is quite flexible and allows to generate DNA, RNA, amino acids or binary sequences. It is possible to give a pml object as input and simSeq return a phyDat from these model. There is also a more low level version, which lacks rate variation, but one can combine different alignments having their own rate (see example).

Value

simSeq returns an object of class phyDat.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

phyDat, pml, SOWH.test

Examples

```r
## Not run:
data(Laurasiatherian)
tree = nj(dist.ml(Laurasiatherian))
fit = pml(tree, Laurasiatherian, k=4)
fit = optim.pml(fit, optNni=TRUE, model="GTR", optGamma=TRUE)
data = simSeq(fit)

## End(Not run)

tree = rtree(5)
plot(tree)
modelabels()

# Example for simple DNA alignment
data = simSeq(tree, l = 10, type="DNA", bf=c(.1,.2,.3,.4), Q=1:6)
as.character(data)

# Example to simulate discrete Gamma rate variation
rates = phangorn:::discrete.gamma(1,4)
data1 = simSeq(tree, l = 100, type="AA", model="WAG", rate=rates[1])
data2 = simSeq(tree, l = 100, type="AA", model="WAG", rate=rates[2])
data3 = simSeq(tree, l = 100, type="AA", model="WAG", rate=rates[3])
```
**SOWH.test**

```r
data4 = simSeq(tree, l = 100, type="AA", model="WAG", rate=rates[4])
data <- c(data1, data2, data3, data4)

write.phyDat(data, file="temp.dat", format="sequential", nbcol = -1, colsep = ")
unlink("temp.dat")
```

---

**Sowford-Olsen-Waddell-Hillis Test**

**Description**

This function computes the Swofford–Olsen–Waddell–Hillis (SOWH) test, a parametric bootstrap test. The function is computationally very demanding and likely to be very slow.

**Usage**

```r
SOWH.test(x, n = 100, restricted = list(optNni=FALSE), optNni=TRUE, trace = 1, ...)
```

**Arguments**

- `x` an object of class "pml".
- `n` the number of bootstrap replicates.
- `restricted` list of restricted parameter settings.
- `optNni` Logical value indicating whether topology gets optimized (NNI).
- `trace` Show output during computations.
- `...` Further arguments passed to "optim.pml".

**Details**

SOWH.test performs a parametric bootstrap test to compare two trees. It makes extensive use of `simSeq` and `optim.pml` and can take quite long.

**Value**

an object of class SOWH. That is a list with three elements, one is a matrix containing for each bootstrap replicate the (log-) likelihood of the restricted and unrestricted estimate and two pml objects of the restricted and unrestricted model.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**


splitsNetwork

Description

splitsNetwork estimates weights for a splits graph from a distance matrix.

Usage

splitsNetwork(dm, splits=NULL, gamma=.1, lambda=1e-6, weight=NULL)

Arguments

dm A distance matrix.
splits a splits object, containing all splits to consider, otherwise all possible splits are used
gamma penalty value for the L1 constraint.
lambda penalty value for the L2 constraint.
weight a vector of weights.

Details

splitsNetwork fits non-negative least-squares phylogenetic networks using L1 (LASSO), L2 (ridge regression) constraints. The function minimizes the penalized least squares

$$\beta = \min \sum (dm - X \beta)^2 + \lambda \| \beta \|_2^2$$
with respect to

\[ \| \beta \|_1 \leq \gamma, \beta > 0 \]

where \( X \) is a design matrix constructed with designSplits. External edges are fitted without L1 or L2 constraints.

**Value**

splitsNetwork returns a splits object with a matrix added. The first column contains the indices of the splits, the second column an unconstrained fit without penalty terms and the third column the constrained fit.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**


**See Also**

distanceHadamard, designTree consensusNet, plot.networx

**Examples**

data(yeast)
dm = dist.ml(yeast)
fit = splitsNetwork(dm)
net = as.networx(fit)
plot(net)
write.nexus.splits(fit)

---

**superTree**

*Super Tree and Species Tree methods*

**Description**

These function superTree allows the estimation of a rooted supertree from a set of rooted trees using Matrix representation parsimony. coalSpeciesTree estimates species trees and can multiple individuals per species.

**Usage**

````
superTree(tree, method = "optim.parsimony", rooted=TRUE, ...)
coalSpeciesTree(tree, X, sTree = NULL)
```
Arguments

- **tree**: an object of class `multiphylo`
- **method**: An argument defining which algorithm is used to optimize the tree.
- **rooted**: should the resulting supertrees be rooted.
- **X**: A phyDat object to define which individual belongs to which species.
- **sTree**: A species tree which forces the topology.
- **...**: further arguments passed to or from other methods.

Details

The function `superTree` extends the function `mrp.supertree` from Liam Revells, with artificial adding an outgroup on the root of the trees. This allows to root the supertree afterwards. The functions is internally used in DensiTree.

cosSpeciesTree estimates a single linkage tree as suggested by Liu et al. (2010) from the element wise minima of the cophenetic matrices of the gene trees. It extends speciesTree in ape as it allows that have several individuals per gene tree.

Value

The function returns an object of class `phylo`.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com> Liam Revell Emmanuel Paradies

References


See Also

`mrp.supertree`, `speciesTree`, `densiTree`, `hclust`

Examples

data(Laurasiatherian)
set.seed(1)
bs <- bootstrap.phyDat(Laurasiatherian, FUN = function(x)upgma(dist.hamming(x)), bs=50)
class(bs) <- 'multiPhylo'
class(bs)
plot(superTree(bs))
**treedist**

*Distances between trees*

**Description**

`treedist` computes different tree distance methods and `RF.dist` the Robinson-Foulds or symmetric distance.

**Usage**

```r
treedist(tree1, tree2, check.labels = TRUE)
RF.dist(tree1, tree2=NULL, check.labels=TRUE)
```

**Arguments**

- `tree1` A phylogenetic tree (class `phylo`) or vector of trees (an object of class `multiphylo`). See details
- `tree2` A phylogenetic tree.
- `check.labels` compares labels of the trees.

**Details**

The Robinson-Foulds distance is well defined only for bifurcating trees. `RF.dist` returns the Robinson-Foulds distance between either 2 trees or computes a matrix of all pairwise distances if a `multiphylo` object is given. For large number of trees `RF.dist` can use a lot of memory!

**Value**

`treedist` returns a vector containing the following tree distance methods

- `symmetric.difference` symmetric.difference or Robinson-Foulds distance
- `branch.score.difference` branch.score.difference
- `path.difference` path.difference
- `weighted.path.difference` weighted.path.difference

**Author(s)**

Klaus P. Schliep <klaus.schliep@gmail.com>

**References**

upgma

UPGMA and WPGMA

Description

UPGMA and WPGMA clustering. Just a wrapper function around hclust.

Usage

upgma(D, method = "average", ...)
wpgma(D, method = "mcquitty", ...)

Arguments

D
  A distance matrix.
metho d
  The agglomeration method to be used. This should be (an unambiguous abbrevi-
  ation of) one of "ward", "single", "complete", "average", "mcquitty", "median"
  or "centroid". The default is "average".
...
  Further arguments passed to or from other methods.

Value

A phylogenetic tree of class phylo.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

hclust, dist.hamming, NJ, as.phylo, fastme, nnls.tree

Examples

data(Laurasiatherian)
dm = dist.ml(Laurasiatherian)
tree = upgma(dm)
plot(tree)
Description
Alignment of 106 genes of 8 different species of yeast.

Usage
data(yeast)

References

Examples
data(yeast)
str(yeast)
Index

*Topic IO
  read.aa, 47
*Topic ancestral
    ancestral.pml, 5
*Topic ancestral
    ancestral.pml, 5
  bab, 8
*Topic classif
  dfactorial, 16
treedist, 55
*Topic cluster
  allTrees, 3
  as.splits, 6
  bab, 8
  bootstrap.pml, 9
designTree, 15
dist.hamming, 17
dist.p, 18
distanceHadamard, 20
getClans, 21
hadamard, 23
lento, 25
midpoint, 26
modelTest, 27
NJ, 30
nni, 31
parsimony, 32
phyDat, 33
pml, 37
pml.fit, 40
pmlCluster, 42
pmlMix, 43
pmlPart, 45
simSeq, 49
splitsNetwork, 52
superTree, 53
upgma, 56
*Topic datasets
  chloroplast, 11

Laurasiatherian, 25
yeast, 57
*Topic hplot
  consensusNet, 12
  neighborNet, 29
*Topic misc
  Ancestors, 4
*Topic models
  SH.test, 48
  SOWH.test, 51
*Topic package
  phangorn-package, 2
*Topic plot
  cladePar, 11
densiTree, 13
lento, 25
plot.networx, 35

acctran (parsimony), 32
acgt2ry (phyDat), 33
addConfidences (as.splits), 6
addTrivialSplits (as.splits), 6
allSitePattern (phyDat), 33
allSplits (as.splits), 6
allTrees, 3
Ancestors, 4
ancestral.pars, 33
ancestral.pars (ancestral.pml), 5
ancestral.pml, 5, 33
anova, 28
as.character.phyDat (phyDat), 33
as.data.frame.phyDat (phyDat), 33
as.DNAbin, 35
as.DNAbin.phyDat (phyDat), 33
as.igraph, 37
as.Matrix (as.splits), 6
as.matrix.splits (as.splits), 6
as.networx, 7
as.networx (plot.networx), 35
as.phyDat (phyDat), 33
INDEX

as.phylo, 56
as.phylo.splits (as.splits), 6
as.prop.part.splits (as.splits), 6
as.splits, 6, 26

bab, 8, 33
baseFreq (phyDat), 33
bootstrap.phyDat (bootstrap.pml), 9
bootstrap.pml, 9, 33, 40
BranchAndBound (bab), 8

Children (Ancestors), 4
chloroplast, 11
CI, 22
CI (parsimony), 32
cladepar, 11
coalSpeciesTree (superTree), 53
compatible (as.splits), 6
consensus, 27
consensusNet, 10, 12, 29, 37, 53
countCycles (as.splits), 6
densiTree, 13, 37, 54
Descendants (Ancestors), 4
designSplits (designTree), 15
designTree, 15, 53
dfactorial, 9, 16
di2multi, 27
dist.dna, 18, 19, 30
dist.hamming, 17, 19, 30, 56
dist.logDet (dist.hamming), 17
dist.ml (dist.hamming), 17
dist.p, 18
distanceHadamard, 7, 13, 15, 20, 23, 24, 37, 53
diversity (getClans), 21
dNAbin, 35
edQt (pml.fit), 40
evonet, 37

factorial, 16
fastme, 15, 30, 56
fhm (hadamard), 23
fitch (parsimony), 32

getClans, 21, 33
getClips (getClans), 21
getDiversity (getClans), 21
getRoot (midpoint), 26

getSlices (getClans), 21
h2st (hadamard), 23
h4st (hadamard), 23
hadamard, 20, 23, 26, 37
hclust, 54, 56

Laurasiatherian, 25
layout.kamada.kawai, 37
ldfactorial (dfactorial), 16
lento, 7, 13, 20, 24, 25, 29
lli (pml.fit), 40

midpoint, 26
modelTest, 27, 40
mrca.phylo (Ancestors), 4

neighborNet, 13, 29, 37
NJ, 30, 33, 56
nj, 30
nni, 31, 33
nnls.networx (designTree), 15
nnls.phylo (designTree), 15
nnls.splits (designTree), 15
nnls.tree, 56
nnls.tree (designTree), 15

optim.parsimony (parsimony), 32
optim.pml, 10
optim.pml (pml), 37
pace (ancestral.pml), 5
parsimony, 22, 32
phangorn (phangorn-package), 2
phangorn-package, 2
phyDat, 22, 33, 48, 50
plot.networx, 13, 14, 20, 24, 29, 35, 53
plot.phylo, 10, 12, 14, 40
plotAnc (ancestral.pml), 5
plotBS (bootstrap.pml), 9
pml, 10, 28, 33, 37, 42–44, 46, 49, 50, 52
pml.fit, 40
pml.free (pml.fit), 40
pml.init (pml.fit), 40
pmlcluster, 42, 44, 46, 49, 52
pmlmix, 35, 40, 42, 43, 46
pmlPart, 40, 42–44, 45, 49, 52
pmlPen (pmlmix), 43
PNJ (parsimony), 32
pratchet, 9
pratchet (parsimony), 32
presenceAbsence (as.splits), 6
print.splits (as.splits), 6
prop.part, 7
pruneTree (midpoint), 26

random.addition (parsimony), 32
read.aa, 35, 47
read.alignment, 48
read.dna, 35, 48
read.GenBank, 48
read.nexus.data, 35
read.nexus.splits (as.splits), 6
read.phyDat (phyDat), 33
RF.dist (treedist), 55
RI, 22
RI (parsimony), 32
rNNI (nni), 31
root, 27
rSPR (nni), 31

sankoff (parsimony), 32
SH.test, 40, 43, 46, 48, 52
Siblings (Ancestors), 4
simSeq, 49, 52
SOWH.test, 49, 50, 51
speciesTree, 54
splitsNetwork, 13, 15, 29, 37, 52
subset.phyDat (phyDat), 33
superTree, 53

treedist, 55

UNJ (NJ), 30
upgma, 15, 30, 56

wpgma (upgma), 56
write.nexus.splits (as.splits), 6
write.phyDat (phyDat), 33
write.splits (as.splits), 6

yeast, 57