

Package ‘phangorn’

November 1, 2017

Title Phylogenetic Reconstruction and Analysis

Version 2.3.1

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

Description Package contains methods for estimation of phylogenetic trees and networks using Maximum Likelihood, Maximum Parsimony, distance methods and Hadamard conjugation. Allows to compare trees, models selection and offers visualizations for trees and split networks.

Depends R (>= 3.2.0), ape (>= 5.0)

Imports quadprog, igraph (>= 1.0), Matrix, parallel, methods, utils, stats, graphics, grDevices, fastmatch, magrittr, Rcpp (>= 0.12.0)

LinkingTo Rcpp

Suggests testthat, seqLogo, seqinr, xtable, flashClust, rgl, knitr, rmarkdown, Biostrings

ByteCompile TRUE

License GPL (>= 2)

VignetteBuilder utils, knitr

URL <https://github.com/KlausVigo/phangorn>

BugReports <https://github.com/KlausVigo/phangorn/issues>

Repository CRAN

RoxygenNote 6.0.1

NeedsCompilation yes

Author Klaus Schliep [aut, cre],
Emmanuel Paradis [aut],
Leonardo de Oliveira Martins [aut],
Alastair Potts [aut],
Tim W. White [aut],
Cyrill Stachniss [ctb],
Michelle Kendall [ctb]

Date/Publication 2017-11-01 16:37:56 UTC

R topics documented:

| | |
|-------------------------------|----|
| phangorn-package | 3 |
| allSplits | 4 |
| allTrees | 6 |
| ancestral.pml | 7 |
| as.network | 8 |
| bab | 11 |
| bootstrap.pml | 12 |
| chloroplast | 14 |
| CI | 15 |
| cladePar | 15 |
| coalSpeciesTree | 16 |
| consensusNet | 17 |
| cophenetic.network | 18 |
| createLabel | 19 |
| delta.score | 20 |
| densiTree | 21 |
| designTree | 23 |
| discrete.gamma | 25 |
| dist.hamming | 26 |
| dist.p | 28 |
| distanceHadamard | 30 |
| fitch | 31 |
| getClans | 33 |
| getRoot | 35 |
| hadamard | 37 |
| identify.network | 38 |
| Laurasiatherian | 39 |
| ldfactorial | 40 |
| lento | 41 |
| mast | 42 |
| maxCladeCred | 43 |
| modelTest | 44 |
| multiphyDat2pmlPart | 46 |
| neighborNet | 47 |
| NJ | 49 |
| nni | 50 |
| phyDat | 51 |
| pml.control | 54 |
| pmlCluster | 57 |
| pmlMix | 59 |
| read.aa | 61 |
| read.nexus.splits | 62 |
| SH.test | 64 |
| simSeq | 65 |
| SOWH.test | 67 |
| splitsNetwork | 68 |

| | |
|-------------------------|-----------|
| <i>phangorn-package</i> | 3 |
| superTree | 70 |
| treedist | 71 |
| upgma | 73 |
| writeDist | 74 |
| yeast | 75 |
| Index | 77 |

| | |
|------------------|-----------------------------------|
| phangorn-package | <i>Phylogenetic analysis in R</i> |
|------------------|-----------------------------------|

Description

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

Details

The complete list of functions can be displayed with `library(help = phangorn)`.
Further information is available in several vignettes. To show the available vignettes in an HTML browser type `browseVignettes("phangorn")`.

| | |
|-------------------|---|
| Trees | Constructing phylogenetic trees (source, pdf) |
| phangorn-specials | Advanced features (source, pdf) |
| Ancestral | Ancestral sequence reconstruction (source, pdf) |
| Networx | Splits and Networx (source, html) |

The first vignette (to display type `vignette('Trees')`) gives an introduction in phylogenetic analysis with phangorn. The second vignette (phangorn-special) covers more advanced feature like defining special character spaces and things which fit nowhere else. More information on **phangorn** can be found on <http://www.phangorn.org>.

Author(s)

Klaus Schliep
Maintainer: Klaus Schliep <klaus.schliep@gmail.com>

References

Schliep K.P. (2011) phangorn: Phylogenetic analysis in R. *Bioinformatics*, 27(4) 592-593

| | |
|-----------|---|
| allSplits | <i>Splits representation of graphs and trees.</i> |
|-----------|---|

Description

as.splits produces a list of splits or bipartitions.

Usage

```
allSplits(k, labels = NULL)

allCircularSplits(k, labels = NULL)

as.splits(x, ...)

## S3 method for class 'splits'
as.matrix(x, zero.print = 0L, one.print = 1L, ...)

## S3 method for class 'splits'
as.Matrix(x, ...)

## S3 method for class 'splits'
print(x, maxp = getOption("max.print"), zero.print = ".",
      one.print = "|", ...)

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

## S3 method for class 'splits'
unique(x, incomparables = FALSE, unrooted = TRUE, ...)

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

## S3 method for class 'multiPhylo'
as.splits(x, ...)

## S3 method for class 'networx'
as.splits(x, ...)

## S3 method for class 'splits'
as.prop.part(x, ...)

## S3 method for class 'splits'
as.bitsplits(x)

compatible(obj)
```

Arguments

| | |
|---------------|---|
| k | number of taxa. |
| labels | names of taxa. |
| x | An object of class phylo or multiPhylo. |
| ... | Further arguments passed to or from other methods. |
| zero.print | character which should be printed for zeros. |
| one.print | character which should be printed for ones. |
| maxp | integer, default from options(max.print), influences how many entries of large matrices are printed at all. |
| recursive | logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector. |
| incomparables | only for compatibility so far. |
| unrooted | todo. |
| obj | an object of class splits. |

Value

as.splits returns an object of class splits, which is mainly a list of splits and some attributes. Often a splits object will contain attributes confidences for bootstrap or Bayesian support values and weight storing edge weights. compatible return a lower triangular matrix where an 1 indicates that two splits are incompatible.

Note

The internal representation is likely to change.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

[prop.part](#), [lento](#), [as.networx](#), [distanceHadamard](#), [read.nexus.splits](#)

Examples

```
(sp <- as.splits(rtree(5)))  
write.nexus.splits(sp)  
spl <- allCircularSplits(5)  
plot(as.networx(spl), "2D")
```

| | |
|----------|--------------------------------------|
| allTrees | <i>Compute all trees topologies.</i> |
|----------|--------------------------------------|

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>

See Also

[rtree](#), [nni](#)

Examples

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

| | |
|---------------|--|
| ancestral.pml | <i>Ancestral character reconstruction.</i> |
|---------------|--|

Description

Marginal reconstruction of the ancestral character states.

Usage

```
ancestral.pml(object, type = "marginal", return = "prob")

ancestral.pars(tree, data, type = c("MPR", "ACCTRAN"), cost = NULL,
  return = "prob")

pace(tree, data, type = c("MPR", "ACCTRAN"), cost = NULL, return = "prob")

plotAnc(tree, data, i = 1, site.pattern = TRUE, col = NULL,
  cex.pie = par("cex"), pos = "bottomright", ...)
```

Arguments

| | |
|--------------|--|
| object | an object of class pml |
| type | method used to assign characters to internal nodes, see details. |
| return | return a phyDat object or matrix of probabilities. |
| tree | a tree, i.e. an object of class pml |
| data | an object of class phyDat |
| cost | A cost matrix for the transitions between two states. |
| i | plots the i-th site pattern of the data. |
| site.pattern | logical, plot i-th site pattern or i-th site |
| 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 |
| ... | 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

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

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

- Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland.
- Swofford, D.L., Maddison, W.P. (1987) Reconstructing ancestral character states under Wagner parsimony. *Math. Biosci.* **87**: 199–229
- Yang, Z. (2006). *Computational Molecular evolution*. Oxford University Press, Oxford.

See Also

[pml](#), [parsimony](#), [ace](#), [root](#)

Examples

```
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)
# plot the first site pattern
plotAnc(tree, anc.ml, 1)
# plot the third character
plotAnc(tree, anc.ml, attr(anc.ml, "index")[3])
```

as.networkx

Phylogenetic networks

Description

as.networkx convert splits objects into a networkx object. And most important there exists a generic plot function to plot phylogenetic network or split graphs.

Usage

```
as.networkx(x, ...)

## S3 method for class 'splits'
as.networkx(x, planar = FALSE, coord = c("none", "2D",
    "3D"), ...)
```



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

## S3 method for class 'networkx'
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 = "black", edge.color = "black", edge.width = 3,
     edge.lty = 1, split.color = NULL, split.width = NULL,
     split.lty = NULL, font = 3, cex = par("cex"), cex.node.label = cex,
     cex.edge.label = cex, col.node.label = tip.color,
     col.edge.label = tip.color, font.node.label = font,
     font.edge.label = font, ...)
```

Arguments

| | |
|-----------------|---|
| x | an object of class "splits" (as.networkx) or "networkx" (plot) |
| ... | Further arguments passed to or from other methods. |
| planar | logical whether to produce a planar graph from only cyclic splits (may excludes splits). |
| coord | add coordinates of the nodes, allows to reproduce the plot. |
| 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. |
| split.color | the colors used to draw edges. |
| split.width | the width used to draw edges. |
| split.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.
`cex.node.label` a numeric value giving the factor scaling of the node labels.
`cex.edge.label` a numeric value giving the factor scaling of the edge labels.
`col.node.label` the colors used for the node labels.
`col.edge.label` the colors used for the edge labels.
`font.node.label`
the font used for the node labels.
`font.edge.label`
the font used for the edge labels.

Details

A `networkx` object hold the information for a phylogenetic network and extends the `phylo` object. Therefore some generic function for `phylo` objects will also work for `networkx` objects. The argument `planar = TRUE` 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 the `rgl` "3D" and basic graphic "2D" device.

Often it is easier and safer to supply vectors of graphical parameters for splits (e.g. `splits.color`) than for edges. These overwrite values `edge.color`.

Note

The internal representation is likely to change.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Dress, A.W.M. and Huson, D.H. (2004) Constructing Splits Graphs *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, **1**(3), 109–115

Schliep, K., Potts, A. J., Morrison, D. A. and Grimm, G. W. (2017), Intertwining phylogenetic trees and networks. *Methods Ecol Evol.* **8**, 1212–1220. doi:10.1111/2041-210X.12760

See Also

[consensusNet](#), [neighborNet](#), [splitsNetwork](#), [hadamard](#), [distanceHadamard](#), [layout_with_kk](#), [evonet](#), [as.igraph](#), [densiTree](#)

Examples

```

set.seed(1)
tree1 <- rtree(20, rooted=FALSE)
sp <- as.splits(rNNI(tree1, n=10))
net <- as.networkx(sp)
plot(net, "2D")

```

```
## Not run:  
# also see example in consensusNet  
example(consensusNet)  
  
## End(Not run)
```

bab*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

- Hendy, M.D. and Penny D. (1982) Branch and bound algorithms to determine minimal evolutionary trees. *Math. Biosc.* **59**, 277-290
- Holland, B.R., Huber, K.T. Penny, D. and Moulton, V. (2005) The MinMax Squeeze: Guaranteeing a Minimal Tree for Population Data, *Molecular Biology and Evolution*, **22**, 235–242
- White, W.T. and Holland, B.R. (2011) Faster exact maximum parsimony search with XMP. *Bioinformatics*, **27(10)**, 1359–1367

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)
```

bootstrap.pml

Bootstrap

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,
             mc.cores = NULL, ...)

bootstrap.phyDat(x, FUN, bs = 100, multicore = FALSE, mc.cores = NULL,
               jumble = TRUE, ...)

plotBS(tree, BStrees, type = "unrooted", bs.col = "black", bs.adj = NULL,
       p = 50, frame = "none", ...)
```

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, whether models should estimated in parallel. |
| 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. |
| jumble | logical, jumble the order of the sequences. |
| 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. |
| p | only plot support values higher than this percentage number (default is 80). |
| frame | a character string specifying the kind of frame to be printed around the bootstrap values. This must be one of "none" (the default), "rect" or "circle". |

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 phylo with the bootstrap values as node labels. The argument BStrees is optional and if not supplied the tree with labels supplied in the node.label slot.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

- Felsenstein J. (1985) Confidence limits on phylogenies. An approach using the bootstrap. *Evolution* **39**, 783–791
- Penny D. and Hendy M.D. (1985) Testing methods evolutionary tree construction. *Cladistics* **1**, 266–278
- Penny D. and Hendy M.D. (1986) Estimating the reliability of evolutionary trees. *Molecular Biology and Evolution* **3**, 403–417

See Also

[optim.pml](#), [pml](#), [plot.phylo](#), [nodelabels](#), [consensusNet](#) and [SOWH.test](#) for parametric bootstrap

Examples

```
## Not run:
data(Laurasiatherian)
dm <- dist.logDet(Laurasiatherian)
tree <- NJ(dm)
# NJ
set.seed(123)
NJtrees <- bootstrap.phyDat(Laurasiatherian,
  FUN=function(x)NJ(dist.logDet(x)), bs=100)
treeNJ <- plotBS(tree, NJtrees, "phylogram")

# Maximum likelihood
fit <- pml(tree, Laurasiatherian)
fit <- optim.pml(fit, rearrangements="NNI")
set.seed(123)
bs <- bootstrap.pml(fit, bs=100, optNni=TRUE)
treeBS <- plotBS(fit$tree,bs)

# Maximum parsimony
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()

# export tree with bootstrap values as node labels
# write.tree(treeBS)

## End(Not run)
```

chloroplast

Chloroplast alignment

Description

Amino acid alignment of 15 genes of 19 different chloroplast.

Examples

```
data(chloroplast)
chloroplast
```

| | |
|----|--|
| CI | <i>Consistency Index and Retention Index</i> |
|----|--|

Description

CI and RI compute the Consistency Index (CI) and Retention Index (RI).

Usage

```
CI(tree, data, cost = NULL, sitewise = FALSE)
```

```
RI(tree, data, cost = NULL, sitewise = FALSE)
```

Arguments

| | |
|----------|---|
| tree | tree to start the nni search from. |
| data | A object of class phyDat containing sequences. |
| cost | A cost matrix for the transitions between two states. |
| sitewise | return CI/RI for alignment or sitewise |

Details

The Consistency Index is defined as minimum number of changes divided by the number of changes required on the tree (parsimony score). The Consistency Index is equal to one if there is no homoplasy. And the Retention Index is defined as

$$RI = \frac{MaxChanges - ObsChanges}{MaxChanges - MinChanges}$$

See Also

[parsimony](#), [pratchet](#), [fitch](#), [sankoff](#), [bab](#), [ancestral.pars](#)

| | |
|----------|---------------------------------------|
| cladePar | <i>Utility function to plot.phylo</i> |
|----------|---------------------------------------|

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)
```

coalSpeciesTree

Species Tree

Description

coalSpeciesTree estimates species trees and can handle multiple individuals per species.

Usage

```
coalSpeciesTree(tree, X = NULL, sTree = NULL)
```


Arguments

| | |
|-------|--|
| tree | an object of class multiPhylo |
| X | A phyDat object to define which individual belongs to which species. |
| sTree | A species tree which fixes the topology. |

Details

coalSpeciesTree 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> Emmanuel Paradies

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.

See Also

[speciesTree](#)

| | |
|--------------|---|
| consensusNet | <i>Computes a consensusNetwork from a list of trees Computes a networkx object from a collection of splits.</i> |
|--------------|---|

Description

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

Usage

```
consensusNet(obj, prob = 0.3, ...)
```

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 networkx. This is just an intermediate to plot phylogenetic networks with igraph.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Holland B.R., Huber K.T., Moulton V., Lockhart P.J. (2004) Using consensus networks to visualize contradictory evidence for species phylogeny. *Molecular Biology and Evolution*, **21**, 1459–61

See Also

[splitsNetwork](#), [neighborNet](#), [lento](#), [distanceHadamard](#), [plot.networkx](#), [maxCladeCred](#)

Examples

```
data(Laurasiatherian)
set.seed(1)
bs <- bootstrap.phyDat(Laurasiatherian, FUN = function(x)nj(dist.hamming(x)),
  bs=50)
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)

tmpfile <- normalizePath(system.file("extdata/trees/RAXML_bootstrap.woodmouse", package="phangorn"))
trees <- read.tree(tmpfile)
cnet_woodmouse <- consensusNet(trees, .3)
plot(cnet_woodmouse, type = "2D", show.edge.label=TRUE)

## End(Not run)
```

cophenetic.networkx

Pairwise Distances from a Phylogenetic Network

Description

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

Usage

```
## S3 method for class 'networkx'
cophenetic(x)
```

Arguments

x an object of class networkx.

Value

an object of class dist, names are set according to the tip labels (as given by the element `tip.label` of the argument `x`).

Author(s)

Klaus Schliep

See Also

[cophenetic](#) for the generic function, `neighborNet` to construct a network from a distance matrix

createLabel

Compare splits and add support values to an object

Description

Add support values to a `splits`, `phylo` or `networkx` object.

Usage

```
createLabel(x, y, label_y, type = "edge", nomatch = NA)

addConfidences(x, y, ...)

presenceAbsence(x, y)
```

Arguments

x an object of class `splits`, `phylo` or `networkx`

y an object of class `splits`, `phylo`, `multiPhylo` or `networkx`

label_y label of y matched on x. Will be usually of `length(as.splits(x))`.

type should labels returned for edges (in `networkx`) or `splits`.

nomatch default value if no match between x and y is found.

... Further arguments passed to or from other methods.

Value

The object `x` with added bootstrap / MCMC support values.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Schliep, K., Potts, A. J., Morrison, D. A. and Grimm, G. W. (2017), Intertwining phylogenetic trees and networks. *Methods Ecol Evol.* **8**, 1212–1220. doi:10.1111/2041-210X.12760

See Also

[as.splits](#), [as.networkx](#), [RF.dist](#), [plot.phylo](#)

Examples

```
data(woodmouse)
woodmouse <- phyDat(woodmouse)
tmpfile <- normalizePath(system.file("extdata/trees/RAXML_bootstrap.woodmouse", package="phangorn"))
boot_trees <- read.tree(tmpfile)

dm <- dist.ml(woodmouse)
tree <- upgma(dm)
nnet <- neighborNet(dm)

tree <- addConfidences(tree, boot_trees)
nnet <- addConfidences(nnet, boot_trees)

plot(tree, show.node.label=TRUE)
plot(nnet, "2D", show.edge.label=TRUE)
```

delta.score

Computes the δ score

Description

Computes the treelikeness

Usage

```
delta.score(x, arg = "mean", ...)
```

Arguments

x an object of class phyDat
 arg Specifies the return value, one of "all", "mean" or "sd"
 ... further arguments passed through `dist.hamming`

Value

A vector containing the δ scores.

Author(s)

Alastair Potts and Klaus Schliep

References

BR Holland, KT Huber, A Dress, V Moulton (2002) δ Plots: a tool for analyzing phylogenetic distance data Russell D. Gray, David Bryant, Simon J. Greenhill (2010) On the shape and fabric of human history *Molecular Biology and Evolution*, **19(12)** 2051–2059
 Russell D. Gray, David Bryant, Simon J. Greenhill (2010) On the shape and fabric of human history *Phil. Trans. R. Soc. B*, **365** 3923–3933; DOI: 10.1098/rstb.2010.0162

See Also

[dist.hamming](#)

Examples

```
data(yeast)
hist(delta.score(yeast, "all"))
```

densiTree

Plots a densiTree.

Description

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

Usage

```
densiTree(x, type = "cladogram", alpha = 1/length(x), consensus = NULL,
  direction = "rightwards", optim = FALSE, scaleX = FALSE, col = 1,
  width = 1, lty = 1, cex = 0.8, font = 3, tip.color = 1, adj = 0,
  srt = 0, underscore = FALSE, label.offset = 0, scale.bar = TRUE,
  jitter = list(amount = 0, random = TRUE), ...)
```

Arguments

| | |
|---------------------------|--|
| <code>x</code> | an object of class <code>multiPhylo</code> . |
| <code>type</code> | a character string specifying the type of phylogeny, so far "cladogram" (default) or "phylogram" are supported. |
| <code>alpha</code> | parameter for semi-transparent colors. |
| <code>consensus</code> | A tree or character vector which is used to define the order of the tip labels. |
| <code>direction</code> | a character string specifying the direction of the tree. Four values are possible: "rightwards" (the default), "leftwards", "upwards", and "downwards". |
| <code>optim</code> | not yet used. |
| <code>scaleX</code> | scale trees to have identical heights. |
| <code>col</code> | a skalar or vector giving the colours used to draw the edges for each plotted phylogeny. These are taken to be in the same order than input trees <code>x</code> . If fewer colours are given than the number of trees, then the colours are recycled. |
| <code>width</code> | edge width. |
| <code>lty</code> | line type. |
| <code>cex</code> | a numeric value giving the factor scaling of the tip labels. |
| <code>font</code> | an integer specifying the type of font for the labels: 1 (plain text), 2 (bold), 3 (italic, the default), or 4 (bold italic). |
| <code>tip.color</code> | color of the tip labels. |
| <code>adj</code> | a numeric specifying the justification of the text strings of the labels: 0 (left-justification), 0.5 (centering), or 1 (right-justification). |
| <code>srt</code> | a numeric giving how much the labels are rotated in degrees. |
| <code>underscore</code> | a logical specifying whether the underscores in tip labels should be written as spaces (the default) or left as are (if TRUE). |
| <code>label.offset</code> | a numeric giving the space between the nodes and the tips of the phylogeny and their corresponding labels. |
| <code>scale.bar</code> | a logical specifying whether add scale.bar to the plot. |
| <code>jitter</code> | allows to shift trees. a list with two arguments: the amount of jitter and random or equally spaced (see details below) |
| <code>...</code> | further arguments to be passed to plot. |

Details

If no consensus tree is provided `densiTree` computes a consensus tree, and if the input trees have different labels a `mrp.supertree` as a backbone. This should avoid too many unnecessary crossings of edges. Trees should be rooted, other wise the output may not be visually pleasing. `jitter` shifts trees a bit so that they are not exactly on top of each other. If `amount == 0`, it is ignored. If `random=TRUE` the result of the perputation is `runif(n, -amount, amount)`, otherwise `seq(-amount, amount, length=n)`, where `n <- length(x)`.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

densiTree is inspired from the great [DensiTree](#) program of Remco Bouckaert.
 Remco R. Bouckaert (2010) DensiTree: making sense of sets of phylogenetic trees *Bioinformatics*, **26** (10), 1372-1373.

See Also

[plot.phylo](#), [plot.networkx](#), [jitter](#)

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, type="cladogram", col="blue")
densiTree(bs, type="phylogram", col="green", direction="downwards", width=2)
# plot five trees slightly shifted, no transparent color
densiTree(bs[1:5], type="phylogram", col=1:5, width=2, jitter=
  list(amount=.3, random=FALSE), alpha=1)
## Not run:
# phylograms 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. computes a contrast matrix if the method is "rooted".

Usage

```
designTree(tree, method = "unrooted", sparse = FALSE, ...)

nnls.tree(dm, tree, rooted = FALSE, trace = 1)
```

```
nnls.phylo(x, dm, rooted = FALSE, trace = 0)

nnls.splits(x, dm, trace = 0)

nnls.networx(x, dm)

designSplits(x, splits = "all", ...)
```

Arguments

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

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

```
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)
```


discrete.gamma

*Internal maximum likelihood functions.***Description**

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

Usage

```
discrete.gamma(alpha, k)

lli(data, tree = NULL, ...)

edQt(Q = c(1, 1, 1, 1, 1, 1), bf = c(0.25, 0.25, 0.25, 0.25))

pml.free()

pml.init(data, k = 1L)

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)
```

Arguments

| | |
|---------------------|---|
| <code>alpha</code> | Shape parameter of the gamma distribution. |
| <code>k</code> | Number of intervals of the discrete gamma distribution. |
| <code>data</code> | An alignment, object of class <code>phyDat</code> . |
| <code>tree</code> | A phylogenetic tree, object of class <code>phylo</code> . |
| <code>...</code> | Further arguments passed to or from other methods. |
| <code>Q</code> | A vector containing the lower triangular part of the rate matrix. |
| <code>bf</code> | Base frequencies. |
| <code>shape</code> | Shape parameter of the gamma distribution. |
| <code>levels</code> | The alphabet used e.g. <code>c("a", "c", "g", "t")</code> for DNA |
| <code>inv</code> | Proportion of invariable sites. |
| <code>rate</code> | Rate. |
| <code>g</code> | vector of quantiles (default is <code>NULL</code>) |
| <code>w</code> | vector of probabilities (default is <code>NULL</code>) |
| <code>eig</code> | Eigenvalue decomposition of <code>Q</code> |
| <code>INV</code> | Sparse representation of invariant sites |

| | |
|-------|---|
| ll.0 | default is NULL |
| llMix | default is NULL |
| wMix | default is NULL |
| site | return the log-likelihood or vector of sitewise likelihood values |

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 and are far less forgiving if the import is not what they expect than pml.

Value

pml.fit returns the log-likelihood.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

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

See Also

[pml](#), [pmlPart](#), [pmlMix](#)

| | |
|--------------|--|
| dist.hamming | <i>Pairwise Distances from Sequences</i> |
|--------------|--|

Description

dist.hamming, dist.ml and dist.logDet compute pairwise distances for an object of class phyDat. dist.ml uses DNA / AA sequences to compute distances under different substitution models.

Usage

```
dist.hamming(x, ratio = TRUE, exclude = "none")

dist.ml(x, model = "JC69", exclude = "none", bf = NULL, Q = NULL,
        k = 1L, shape = 1, ...)

dist.logDet(x)
```

Arguments

| | |
|---------|--|
| x | An object of class phyDat |
| ratio | Compute uncorrected ('p') distance or character difference. |
| 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. |
| model | One of "JC69", "F81" or one of 17 amino acid models see details. |
| bf | A vector of base frequencies. |
| Q | A vector containing the lower triangular part of the rate matrix. |
| k | Number of intervals of the discrete gamma distribution. |
| shape | Shape parameter of the gamma distribution. |
| ... | Further arguments passed to or from other methods. |

Details

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

The "F81" model uses empirical base frequencies, the "JC69" equal base frequencies. This is even the case if the data are not nucleotides.

Value

an object of class `dist`

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

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.

Jukes TH and Cantor CR (1969). *Evolution of Protein Molecules*. New York: Academic Press. 21–132.

See Also

For more distance methods for nucleotide data see [dist.dna](#) and [dist.p](#) for pairwise polymorphism p-distances. [writeDist](#) for export and import distances.

Examples

```
data(Laurasiatherian)
dm1 <- dist.hamming(Laurasiatherian)
tree1 <- NJ(dm1)
dm2 <- dist.logDet(Laurasiatherian)
tree2 <- NJ(dm2)
treedist(tree1, tree2)
# JC model
dm3 <- dist.ml(Laurasiatherian)
tree3 <- NJ(dm3)
treedist(tree1, tree3)
# F81 + Gamma
dm4 <- dist.ml(Laurasiatherian, model="F81", k=4, shape=.4)
tree4 <- NJ(dm4)
treedist(tree1, tree4)
treedist(tree3, tree4)
```

dist.p

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 predefined 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. 2014) 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`.

Author(s)

Klaus Schliep and Alastair Potts

References

Alvarez, I., and J. F. Wendel. (2003) Ribosomal ITS sequences and plant phylogenetic inference. *Molecular Phylogenetics and Evolution*, **29**, 417–434.

Bailey, C. D., T. G. Carr, S. A. Harris, and C. E. Hughes. (2003) Characterization of angiosperm nrDNA polymorphism, paralogy, and pseudogenes. *Molecular Phylogenetics and Evolution* **29**, 435–455.

Goeker, M., and G. Grimm. (2008) General functions to transform associate data to host data, and their use in phylogenetic inference from sequences with intra-individual variability. *BMC Evolutionary Biology*, **8**:86.

Potts, A.J., T.A. Hedderson, and G.W. Grimm. (2014) Constructing phylogenies in the presence of intra-individual site polymorphisms (2ISPs) with a focus on the nuclear ribosomal cistron. *Systematic Biology*, **63**, 1–16

See Also

[dist.dna](#), [dist.hamming](#)

Examples

```
data(Laurasiatherian)
laura <- as.DNABin(Laurasiatherian)

dm <- dist.p(Laurasiatherian, "polymorphism")

#####
# 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)
```

| | |
|------------------|--------------------------|
| distanceHadamard | <i>Distance Hadamard</i> |
|------------------|--------------------------|

Description

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

Hendy, M. D. and Penny, D. (1993). Spectral Analysis of Phylogenetic Data. *Journal of Classification*, **10**, 5-24.

See Also

[hadamard](#), [lento](#), [plot.networkx](#), [neighborNet](#)

Examples

```
data(yeast)
dm <- dist.hamming(yeast)
dm <- as.matrix(dm)
fit <- distanceHadamard(dm)
lento(fit)
plot(as.networkx(fit), "2D")
```

| | |
|-------|------------------------|
| fitch | <i>Parsimony tree.</i> |
|-------|------------------------|

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.

Usage

```
fitch(tree, data, site = "pscore")

random.addition(data, method = "fitch")

parsimony(tree, data, method = "fitch", ...)

sankoff(tree, data, cost = NULL, site = "pscore")

optim.parsimony(tree, data, method = "fitch", cost = NULL, trace = 1,
  rearrangements = "SPR", ...)

pratchet(data, start = NULL, method = "fitch", maxit = 1000, k = 10,
  trace = 1, all = FALSE, rearrangements = "SPR",
  perturbation = "ratchet", ...)

acctran(tree, data)
```

Arguments

| | |
|----------------|---|
| tree | tree to start the nni search from. |
| data | A object of class phyDat containing sequences. |
| site | return either 'pscore' or 'site' wise parsimony scores. |
| method | one of 'fitch' or 'sankoff'. |
| ... | Further arguments passed to or from other methods (e.g. model="sankoff" and cost matrix). |
| cost | A cost matrix for the transitions between two states. |
| 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.
 perturbation whether using a ratchet or stochastic (nni) for shuffling the tree.

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

Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland.
 Nixon, K. (1999) The Parsimony Ratchet, a New Method for Rapid Parsimony Analysis. *Cladistics* **15**, 407-414

See Also

[bab](#), [CI](#), [RI](#), [ancestral.pml](#), [nni](#), [NJ](#), [pml](#), [getClans](#), [ancestral.pars](#), [bootstrap.pml](#)

Examples

```
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, maxit=100, k=5)
# assign edge length
treeRatchet <- acctran(treeRatchet, Laurasiatherian)

plot(midpoint(treeRatchet))
add.scale.bar(0,0, length=100)

parsimony(c(tree,treeNNI, treeRatchet), Laurasiatherian)
```

| | |
|----------|--------------------------------|
| getClans | <i>Clans, slices and clips</i> |
|----------|--------------------------------|

Description

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

Usage

```
getClans(tree)

getSlices(tree)

getClips(tree, all = TRUE)

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 leaf).

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 $2n^2 - 10n + 12$.

cophenetic distance and not by the topology. Namely clips are groups of leaves for which the maximum pairwise distance is smaller than threshold.

distance within a clip is lower than the distance between any member of the clip and any other tip.

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 categorical 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: $H = - \sum_{i=1}^k (N_i/N) \log(N_i/N)$, $N = \sum_{i=1}^k N_i$

or the

Equitability Index: $E = H/\log(N)$

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 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

Lapointe, F.-J., Lopez, P., Boucher, Y., Koenig, J., Baptiste, E. (2010) Clanistics: a multi-level perspective for harvesting unrooted gene trees. *Trends in Microbiology* 18: 341-347

Wilkinson, M., McInerney, J.O., Hirt, R.P., Foster, P.G., Embley, T.M. (2007) Of clades and clans: terms for phylogenetic relationships in unrooted trees. *Trends in Ecology and Evolution* 22: 114-115

Schliep, K., Lopez, P., Lapointe F.-J., Baptiste E. (2011) Harvesting Evolutionary Signals in a Forest of Prokaryotic Gene Trees, *Molecular Biology and Evolution* 28(4): 1393-1405

See Also

[parsimony](#), Consistency index [CI](#), Retention index [RI](#), [phyDat](#)

Examples

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

set.seed(123)
trees <- rmtree(10, 20)
X <- matrix(sample(c("red", "blue", "violet"), 100, TRUE, c(.5,.4, .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,1]) # intruders are blue

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

getRoot

Tree manipulation

Description

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

Usage

```
getRoot(tree)

midpoint(tree, node.labels = "support", ...)

## S3 method for class 'phylo'
midpoint(tree, node.labels = "support", ...)

## S3 method for class 'multiPhylo'
midpoint(tree, node.labels = "support", ...)

pruneTree(tree, ..., FUN = ">=")
```

Arguments

| | |
|--------------------------|--|
| <code>tree</code> | an object of class <code>phylo</code> |
| <code>node.labels</code> | are node labels 'support' values, 'label' or should be 'deleted' |
| <code>...</code> | further arguments, passed to other methods. |
| <code>FUN</code> | a function evaluated on the node labels, result must be logical. |

Details

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

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)
```

hadamard

Hadamard Matrices and Fast Hadamard Multiplication

Description

A collection of functions to perform Hadamard conjugation. Hadamard matrix H with a vector v using fast Hadamard multiplication.

Usage

```
hadamard(x)
```

```
fhm(v)
```

```
h4st(obj, levels = c("a", "c", "g", "t"))
```

```
h2st(obj, eps = 0.001)
```

Arguments

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

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 Splitsree.

Value

`hadamard` returns a Hadamard matrix. `fhm` returns the fast Hadamard multiplication.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Hendy, M.D. (1989). The relationship between simple evolutionary tree models and observable sequence data. *Systematic Zoology*, **38** 310–321.

Hendy, M. D. and Penny, D. (1993). Spectral Analysis of Phylogenetic Data. *Journal of Classification*, **10**, 5–24.

Hendy, M. D. (2005). Hadamard conjugation: an analytical tool for phylogenetics. In O. Gascuel, editor, *Mathematics of evolution and phylogeny*, Oxford University Press, Oxford

Waddell P. J. (1995). Statistical methods of phylogenetic analysis: Including hadamard conjugation, LogDet transforms, and maximum likelihood. *PhD thesis*.

See Also

[distanceHadamard](#), [lento](#), [plot.networkx](#)

Examples

```
H <- hadamard(3)
v <- 1:8
H %*% v
fhm(v)

data(yeast)

# RY-coding
dat_ry <- acgt2ry(yeast)
fit2 <- h2st(dat_ry)
lento(fit2)

# write.nexus.splits(fit2, file = "test.nxs")
# read this file into Spectronet or Splitstree to show the network
## Not run:
dat <- as.character(yeast)
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)
```

identify.networkx

Identify splits in a network

Description

identify.networkx reads the position of the graphics pointer when the mouse button is pressed. It then returns the split belonging to the edge closest to the pointer. The network must be plotted beforehand.

Usage

```
## S3 method for class 'networkx'  
identify(x, quiet = FALSE, ...)
```

Arguments

| | |
|-------|--|
| x | an object of class networkx |
| 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. |

Value

identify.networkx returns a splits object.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

[plot.networkx](#)

Examples

```
## Not run:  
data(yeast)  
dm <- dist.ml(yeast)  
nnet <- neighborNet(dm)  
plot(nnet, "2D")  
identify(nnet) # click close to an edge  
  
## End(Not run)
```

Laurasiatherian

Laurasiatherian data (AWCMEE)

Description

Laurasiatherian RNA sequence data

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)
```

| | |
|-------------|-----------------------------|
| ldfactorial | <i>Arithmetic Operators</i> |
|-------------|-----------------------------|

Description

double factorial function

Usage

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

Arguments

x a numeric scalar or vector

Value

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

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

[factorial](#), [howmanytrees](#)

Examples

```
dfactorial(1:10)
```

| | |
|-------|-------------------|
| lento | <i>Lento plot</i> |
|-------|-------------------|

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,  
      col = rgb(0, 0, 0, 0.5), ...)
```

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). |
| col | color for the splits / bipartition. |
| ... | Further arguments passed to or from other methods. |

Value

lento returns a plot.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Lento, G.M., Hickson, R.E., Chambers G.K., and Penny, D. (1995) Use of spectral analysis to test hypotheses on the origin of pinnipeds. *Molecular Biology and Evolution*, **12**, 28-52.

See Also

[as.splits](#), [hadamard](#)

Examples

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

| | |
|------|----------------------------------|
| mast | <i>Maximum agreement subtree</i> |
|------|----------------------------------|

Description

mast computes the maximum agreement subtree (MAST).

Usage

```
mast(x, y, tree = TRUE, rooted = TRUE)
```

Arguments

| | |
|--------|---|
| x | a tree, i.e. an object of class phylo. |
| y | a tree, i.e. an object of class phylo. |
| tree | a logical, if TRUE returns a tree other wise the tip labels of the the maximum agreement subtree. |
| rooted | logical if TRUE treats trees as rooted otherwise unrooted. |

Details

The code is derived from the code example in Valiente (2009), for the original code see <http://www.cs.upc.edu/~valiente/comput-biol/>. The version for the unrooted trees is much slower.

Value

mast returns a vector of the tip labels in the MAST.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com> based on code of Gabriel Valiente

References

G. Valiente (2009). *Combinatorial Pattern Matching Algorithms in Computational Biology using Perl and R*. Taylor & Francis/CRC Press

See Also

[SPR.dist](#)

Examples

```
tree1 <- rtree(100)
tree2 <- rSPR(tree1, 5)
tips <- mast(tree1, tree2)
```

| | |
|--------------|---------------------------------------|
| maxCladeCred | <i>Maximum clade credibility tree</i> |
|--------------|---------------------------------------|

Description

maxCladeCred computes the maximum clade credibility tree from a sample of trees.

Usage

```
maxCladeCred(x, tree = TRUE, part = NULL, rooted = TRUE)
```

Arguments

| | |
|--------|---|
| x | x is an object of class multiPhylo or phylo |
| tree | logical indicating whether return the tree with the clade credibility (default) or the clade credibility score for all trees. |
| part | a list of partitions as returned by prop.part |
| rooted | logical, if FALSE the tree with highest maximum bipartition credibility is returned. |

Details

So far just the best tree is returned. No annotations or transformations of edge length are performed.

If a list of partition is provided then the clade credibility is computed for the trees in x.

Value

a tree (an object of class phylo) with the highest clade credibility or a numeric vector of clade credibilities for each tree.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

[consensus](#), [consensusNet](#), [prop.part](#)

Examples

```
data(Laurasiatherian)
set.seed(42)
bs <- bootstrap.phyDat(Laurasiatherian, FUN = function(x)upgma(dist.hamming(x)),
  bs=100)

strict_consensus <- consensus(bs)
majority_consensus <- consensus(bs, p=.5)
max_clade_cred <- maxCladeCred(bs)
par(mfrow = c(1,3), mar = c(1,4,1,1))
plot(strict_consensus, main="Strict consensus tree")
plot(majority_consensus, main="Majority consensus tree")
plot(max_clade_cred, main="Maximum clade credibility tree")

# compute clade credibility for trees given a prop.part object
pp <- prop.part(bs)
tree <- rNNI(bs[[1]], 20)
maxCladeCred(c(tree, bs[[1]]), tree=FALSE, part = pp)
# first value likely be -Inf
```

modelTest

ModelTest

Description

Comparison of different nucleotide or amino acid substitution models

Usage

```
modelTest(object, tree = NULL, model = c("JC", "F81", "K80", "HKY", "SYM",
  "GTR"), G = TRUE, I = TRUE, FREQ = FALSE, k = 4,
  control = pml.control(epsilon = 1e-08, maxit = 10, trace = 1),
  multicore = FALSE, mc.cores = NULL)
```

Arguments

| | |
|--------|--|
| object | an object of class phyDat or pml |
| tree | a phylogenetic tree. |
| model | a vector containing the substitution models to compare with each other or "all" to test all available models |
| G | logical, TRUE (default) if (discrete) Gamma model should be tested |
| I | logical, TRUE (default) if invariant sites should be tested |
| FREQ | logical, FALSE (default) if TRUE amino acid frequencies will be estimated. |
| k | number of rate classes |

| | |
|-----------|--|
| control | A list of parameters for controlling the fitting process. |
| multicore | logical, whether models should be estimated in parallel. |
| mc.cores | The number of cores to use, i.e. at most how many child processes will be run simultaneously. Must be at least one, and parallelization requires at least two cores. |

Details

modelTest estimates all the specified models for a given tree and data. When the mclapply is available, the computations are done in parallel. modelTest runs each model in one thread. This may not work within a GUI interface and will not work under Windows.

Value

A data.frame containing the log-likelihood, number of estimated parameters, AIC, AICc and BIC all tested models. The data.frame has an attribute "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

- Burnham, K. P. and Anderson, D. R (2002) *Model selection and multimodel inference: a practical information-theoretic approach*. 2nd ed. Springer, New York
- Posada, D. and Crandall, K.A. (1998) MODELTEST: testing the model of DNA substitution. *Bioinformatics* **14**(9): 817-818
- Posada, D. (2008) jModelTest: Phylogenetic Model Averaging. *Molecular Biology and Evolution* **25**: 1253-1256
- Darriba D., Taboada G.L., Doallo R and Posada D. (2011) ProtTest 3: fast selection of best-fit models of protein evolution. *Bioinformatics* **27**: 1164-1165

See Also

[pml](#), [anova](#), [AIC](#)

Examples

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

# some R magic
env <- attr(mT, "env")
ls(env=env)
(F81 <- get("F81+G", env)) # a call
```

```
eval(F81, env=env)

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

# test all available amino acid models
(mTAA_all <- modelTest(chloroplast, model="all", multicore=TRUE, mc.cores=2))

## End(Not run)
```

multiphyDat2pmlPart *Partition model.*

Description

Model to estimate phylogenies for partitioned data.

Usage

```
multiphyDat2pmlPart(x, rooted = FALSE, ...)

pmlPart2multiPhylo(x)

pmlPart(formula, object, control = pml.control(epsilon = 1e-08, maxit = 10,
  trace = 1), model = NULL, rooted = FALSE, ...)
```

Arguments

| | |
|---------|---|
| x | an object of class pmlPart |
| rooted | Are the gene trees rooted (ultrametric) or unrooted. |
| ... | Further arguments passed to or from other methods. |
| 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. |

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, pmlPen can try to find more parsimonious models (see example).

pmlPart2multiPhylo is a convenience function to extract the trees out of a pmlPart object.

Value

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

[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)
```

neighborNet

Computes a neighborNet from a distance matrix

Description

Computes a neighborNet, i.e. an object of class network 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 networkx.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

Bryant, D. & Moulton, V. (2004) Neighbor-Net: An Agglomerative Method for the Construction of Phylogenetic Networks. *Molecular Biology and Evolution*, 2004, **21**, 255-265

See Also

[splitsNetwork](#), [consensusNet](#), [plot.networkx](#), [lento](#), [cophenetic.networkx](#), [distanceHadamard](#)

Examples

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

| | |
|----|-------------------------|
| NJ | <i>Neighbor-Joining</i> |
|----|-------------------------|

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

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*, **6**, 729–731.

Gascuel, O. (1997) Concerning the NJ algorithm and its unweighted version, UNJ. in Birkin et. al. *Mathematical Hierarchies and Biology*, 149–170.

See Also

[nj](#), [dist.dna](#), [dist.hamming](#), [upgma](#), [fastme](#)

Examples

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

| | |
|-----|-----------------------------|
| nni | <i>Tree rearrangements.</i> |
|-----|-----------------------------|

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)

rNNI(tree, moves = 1, n = length(moves))

rSPR(tree, moves = 1, n = length(moves), k = NULL)
```

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>

See Also

[allTrees](#), [SPR.dist](#)

Examples

```
tree <- unroot(rtree(20))
trees1 <- nni(tree)
trees2 <- rSPR(tree, 2, 10)
```

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, ...)

dna2codon(x)

codon2dna(x)

as.phyDat(x, ...)

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

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

## S3 method for class 'alignment'
as.phyDat(x, type = "DNA", ...)

phyDat2alignment(x)

## S3 method for class 'MultipleAlignment'
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'
as.AABin(x, ...)

write.phyDat(x, file, format = "phylip", colsep = "", nbc col = -1, ...)

read.phyDat(file, format = "phylip", type = "DNA", ...)
```

```

baseFreq(obj, freq = FALSE, all = FALSE, drop.unused.levels = FALSE)

## S3 method for class 'phyDat'
subset(x, subset, select, site.pattern = TRUE, ...)

## S3 method for class 'phyDat'
unique(x, incomparables = FALSE, identical = TRUE, ...)

allSitePattern(n, levels = c("a", "c", "g", "t"), names = NULL)

genlight2phyDat(x, ambiguity = NA)

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

```

Arguments

| | |
|--------------------|---|
| data | 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. |
| ... | further arguments passed to or from other methods. |
| x | An object containing sequences. |
| allLevels | return original data. |
| file | A file name. |
| format | File format of the sequence alignment (see details). Several popular formats are supported: "phylip", "interleaved", "sequential", "clustal", "fasta" or "nexus", or any unambiguous abbreviation of these. |
| colsep | a character used to separate the columns (a single space by default). |
| nbcol | a numeric specifying the number of columns per row (-1 by default); may be negative implying that the nucleotides are printed on a single line. |
| obj | as object of class phyDat |
| freq | logical, if 'TRUE', frequencies or counts are returned otherwise proportions |
| all | all a logical; if all = TRUE, all counts of bases, ambiguous codes, missing data, and alignment gaps are returned as defined in the contrast. |
| drop.unused.levels | logical, drop unused levels |
| subset | a subset of taxa. |
| select | a subset of characters. |
| site.pattern | select site pattern or sites. |
| incomparables | for compatibility with unique. |

| | |
|-----------|---|
| identical | if TRUE (default) sequences have to be identical, if FALSE sequences are considered duplicates if distance between sequences is zero (happens frequently with ambiguous sites). |
| n | Number of sequences. |
| names | Names of sequences. |
| ambiguity | character for ambiguous character and no contrast is provided. |

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. There is a more detailed example for specifying "USER" defined data formats in the vignette "phangorn-specials".

`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.

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`, `read.nexus.data` and the chapter 1 in the vignette("phangorn-specials", package="phangorn") and the example of `pmlMix` for the use of `allSitePattern`

Examples

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

```
all.equal(Laurasiatherian, Laura)
allSitePattern(5)
```

pml.control

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

```
pml.control(epsilon = 1e-08, maxit = 10, trace = 1)

pml(tree, data, bf = NULL, Q = NULL, inv = 0, k = 1, shape = 1,
     rate = 1, model = NULL, ...)

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 = 1L), model = NULL, rearrangement = ifelse(optNni, "NNI", "none"),
          subs = NULL, ratchet.par = list(iter = 20L, maxit = 100L, prop = 1/3),
          ...)
```

Arguments

| | |
|---------|---|
| epsilon | Stop criterion for optimisation (see details). |
| maxit | Maximum number of iterations (see details). |
| trace | Show output during optimization (see details). |
| 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 models or nucleotide model, see details. |
| ... | Further arguments passed to or from other methods. |
| object | An object of class pml. |
| optNni | Logical value indicating whether topology gets optimized (NNI). |

| | |
|---------------|--|
| optBf | Logical value indicating whether base frequencies gets 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 the edge lengths gets optimized. |
| optRate | Logical value indicating the overall rate gets optimized. |
| optRooted | Logical value indicating if the edge lengths of a rooted tree get optimized. |
| control | A list of parameters for controlling the fitting process. |
| rearrangement | type of tree tree rearrangements to perform, one of "none", "NNI", "stochastic" or "ratchet" |
| subs | A (integer) vector same length as Q to specify the optimization of Q |
| ratchet.par | search parameter for stochastic search |

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 optimised, 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:

| model | optBf | optQ |
|--------------|-------|-------|
| Jukes-Cantor | FALSE | FALSE |
| F81 | TRUE | FALSE |
| symmetric | FALSE | TRUE |
| GTR | TRUE | TRUE |

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 ("WAG", "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.

It is also possible to estimate codon models (e.g. YN98), for details see also the chapter in vignette("phangorn-specials").

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.

pml.control 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 $(\log\text{Lik}(k) - \log\text{Lik}(k+1)) / \log\text{Lik}(k+1)$, this seems to be a good heuristics which works reasonably 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.

If rearrangement is set to stochastic a stochastic search algorithm similar to Nguyen et al. (2015). and for ratchet the likelihood ratchet as in Vos (2003). This should help often to find better tree topologies, especially for larger trees.

Value

pml or optim.pml return a list of class pml, some are useful for further computations like

| | |
|---------|------------------------------|
| tree | the phylogenetic tree. |
| data | the alignment. |
| logLik | Log-likelihood of the tree. |
| siteLik | Site log-likelihoods. |
| weight | Weight of the site patterns. |

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

References

- Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, **17**, 368–376.
- Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland.
- Yang, Z. (2006). *Computational Molecular evolution*. Oxford University Press, Oxford.
- Adachi, J., P. J. Waddell, W. Martin, and M. Hasegawa (2000) Plastid genome phylogeny and a model of amino acid substitution for proteins encoded by chloroplast DNA. *Journal of Molecular Evolution*, **50**, 348–358
- Rota-Stabelli, O., Z. Yang, and M. Telford. (2009) MtZoa: a general mitochondrial amino acid substitutions model for animal evolutionary studies. *Mol. Phyl. Evol.*, **52**(1), 268–72
- Whelan, S. and Goldman, N. (2001) A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, **18**, 691–699
- Le, S.Q. and Gascuel, O. (2008) LG: An Improved, General Amino-Acid Replacement Matrix *Molecular Biology and Evolution*, **25**(7), 1307–1320
- Yang, Z., R. Nielsen, and M. Hasegawa (1998) Models of amino acid substitution and applications to Mitochondrial protein evolution. *Molecular Biology and Evolution*, **15**, 1600–1611
- Abascal, F., D. Posada, and R. Zardoya (2007) MtArt: A new Model of amino acid replacement for Arthropoda. *Molecular Biology and Evolution*, **24**, 1–5
- Kosiol, C, and Goldman, N (2005) Different versions of the Dayhoff rate matrix - *Molecular Biology and Evolution*, **22**, 193–199
- L.-T. Nguyen, H.A. Schmidt, A. von Haeseler, and B.Q. Minh (2015) IQ-TREE: A fast and effective stochastic algorithm for estimating maximum likelihood phylogenies. *Molecular Biology and Evolution*, **32**, 268–274.
- Vos, R. A. (2003) Accelerated Likelihood Surface Exploration: The Likelihood Ratchet. *Systematic Biology*, **52**(3), 368–373

Yang, Z., and R. Nielsen (1998) Synonymous and nonsynonymous rate variation in nuclear genes of mammals. *Journal of Molecular Evolution*, **46**, 409-418.

Lewis, P.O. (2001) A likelihood approach to estimating phylogeny from discrete morphological character data. *Systematic Biology* **50**, 913-925.

See Also

[bootstrap.pml](#), [modelTest](#), [pmlPart](#), [pmlMix](#), [plot.phylo](#), [SH.test](#), [ancestral.pml](#)

Examples

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

## Not run:
# search for a better tree using NNI rearrangements
fitJC <- optim.pml(fitJC, optNni=TRUE)
fitJC
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, rearrangement = "stochastic",
  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")
```

Description

Stochastic Partitioning of genes into p cluster.

Usage

```
pmlCluster(formula, fit, weight, p = 1:5, part = NULL, nrep = 10,
  control = pml.control(epsilon = 1e-08, 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>

References

K. P. Schliep (2009). Some Applications of statistical phylogenetics (PhD Thesis)
 Lanfear, R., Calcott, B., Ho, S.Y.W. and Guindon, S. (2012) PartitionFinder: Combined Selection of Partitioning Schemes and Substitution Models for Phylogenetic Analyses. *Molecular Biology and Evolution*, **29**(6), 1695-1701

See Also

[pml](#), [pmlPart](#), [pmlMix](#), [SH.test](#)

Examples

```
## 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

```
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 ~ rate + shape + ...{}`, 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

```
## 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))

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

fit2 <- update(fit, k=4)
fit2 <- optim.pml(fit2, optGamma=TRUE)
```

```

fitMix <- pmlMix(edge ~ rate, fit, m=4)
fitMix

#
# 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))
}

## End(Not run)

```

read.aa

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

```

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

<https://www.ncbi.nlm.nih.gov/blast/fasta.shtml> 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](#), [phyDat](#), [read.alignment](#)

read.nexus.splits

Function to import and export splits and networks

Description

read.nexus.splits, write.nexus.splits, read.nexus.networkx, write.nexus.networkx can be used to import and export splits and networks with nexus format and allow to exchange these object with other software like Splitsree. write.splits returns a human readable output.

Usage

```

read.nexus.splits(file)

write.nexus.splits(obj, file = "", weights = NULL, taxa = TRUE,
  append = FALSE)

write.nexus.networkx(obj, file = "", taxa = TRUE, splits = TRUE,
  append = FALSE)

read.nexus.networkx(file, splits = TRUE)

write.splits(x, file = "", zero.print = ".", one.print = "|",
  print.labels = TRUE, ...)

```

Arguments

| | |
|--------------|--|
| file | a file name. |
| obj | An object of class splits. |
| weights | Edge weights. |
| taxa | logical. If TRUE a taxa block is added |
| append | logical. If TRUE the nexus blocks will be added to a file. |
| splits | logical. If TRUE the nexus blocks will be added to a file. |
| x | An object of class splits. |
| zero.print | character which should be printed for zeros. |
| one.print | character which should be printed for ones. |
| print.labels | logical. If TRUE labels are printed. |
| ... | Further arguments passed to or from other methods. |
| labels | names of taxa. |

Value

write.nexus.splits and write.nexus.networkx write out the splits and networkx object to read with other software like SplitsTree. read.nexus.splits and read.nexus.networkx return an splits and networkx object.

Note

read.nexus.splits 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](#), [as.splits](#), [as.networx](#)

Examples

```
(sp <- as.splits(rtree(5)))
write.nexus.splits(sp)
spl <- allCircularSplits(5)
plot(as.networx(spl), "2D")
write.splits(spl, print.labels = FALSE)
```

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

Shimodaira, H. and Hasegawa, M. (1999) Multiple comparisons of log-likelihoods with applications to phylogenetic inference. *Molecular Biology and Evolution*, **16**, 1114–1116.

See Also

[pml](#), [pmlPart](#), [pmlCluster](#), [SOWH.test](#)

Examples

```
data(Laurasiatherian)
dm <- dist.logDet(Laurasiatherian)
tree1 <- NJ(dm)
tree2 <- unroot(upgma(dm))
fit1 <- pml(tree1, Laurasiatherian)
fit2 <- pml(tree2, Laurasiatherian)
fit1 <- optim.pml(fit1) # optimize edge weights
fit2 <- optim.pml(fit2)
SH.test(fit1, fit2, B=500)
# in real analysis use larger B, e.g. 10000
## Not run:
example(pmlPart)
SH.test(sp, B=1000)

## End(Not run)
```

simSeq

Simulate sequences.

Description

Simulate sequences for a given evolutionary tree.

Usage

```
simSeq(x, ...)
```

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

```
## S3 method for class 'pml'
simSeq(x, ancestral = FALSE, ...)
```

Arguments

| | |
|---------|---|
| x | a phylogenetic tree tree, i.e. an object of class phylo or and object of class pml. |
| ... | Further arguments passed to or from other methods. |
| 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: e.g. "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 | mutation rate or scaler for the edge length, a numerical value greater than zero. |
| ancestral | Return ancestral sequences? |

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 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). The rate parameter acts like a scaler for the edge lengths.

Value

simSeq returns an object of class phyDat.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

[phyDat](#), [pml](#), [SOWH.test](#)

Examples

```
## 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)
nodeLabels()

# 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 = 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])
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")

```

SOWH.test

*Swofford-Olsen-Waddell-Hillis Test***Description**

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

Usage

```

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

```

Arguments

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

Details

SOWH.test performs a parametric bootstrap test to compare two trees. It makes extensive use 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

Goldman, N., Anderson, J. P., and Rodrigo, A. G. (2000) Likelihood -based tests of topologies in phylogenetics. *Systematic Biology* **49** 652-670.

Swofford, D.L., Olsen, G.J., Waddell, P.J. and Hillis, D.M. (1996) Phylogenetic Inference in Hillis, D.M., Moritz, C. and Mable, B.K. (Eds.) *Molecular Systematics* (2nd ed.) 407-514, Sunderland, MA: Sinauer

See Also

[pml](#), [pmlPart](#), [pmlCluster](#), [simSeq](#), [SH.test](#)

Examples

```
# in real analysis use larger n, e.g. 500 preferably more
## Not run:
data(Laurasiatherian)
dm <- dist.logDet(Laurasiatherian)
tree <- NJ(dm)
fit <- pml(tree, Laurasiatherian)
fit <- optim.pml(fit, TRUE)
set.seed(6)
tree <- rNNI(fit$tree, 1)
fit <- update(fit, tree = tree)
(res <- SOWH.test(fit, n=100))
summary(res)

## End(Not run)
```

splitsNetwork

Phylogenetic Network

Description

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

Usage

```
splitsNetwork(dm, splits = NULL, gamma = 0.1, lambda = 1e-06,
  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

$$\|\beta\|_1 \leq \gamma, \beta \geq 0$$

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

- Efron, Hastie, Johnstone and Tibshirani (2004) Least Angle Regression (with discussion) *Annals of Statistics* **32**(2), 407–499
- K. P. Schliep (2009). Some Applications of statistical phylogenetics (PhD Thesis)

See Also

[distanceHadamard](#), [designTree](#) [consensusNet](#), [plot.networkx](#)

Examples

```
data(yeast)
dm <- dist.ml(yeast)
fit <- splitsNetwork(dm)
net <- as.networkx(fit)
plot(net, "2D")
write.nexus.splits(fit)
```

superTree

*Super Tree methods***Description**

These function superTree allows the estimation of a supertree from a set of trees using either Matrix representation parsimony, Robinson-Foulds or SPR as criterion.

Usage

```
superTree(tree, method = "MRP", rooted = FALSE, trace = 0, start = NULL,
          multicore = FALSE, mc.cores = NULL, ...)
```

Arguments

| | |
|-----------|--|
| tree | an object of class multiPhylo |
| method | An argument defining which algorithm is used to optimize the tree. Possible are "MRP", "NNI", and "SPR". |
| rooted | should the resulting supertrees be rooted. |
| trace | defines how much information is printed during optimization. |
| start | a starting tree can be supplied. |
| multicore | logical, whether models should estimated in parallel. |
| mc.cores | The number of cores to use, i.e. at most how many child processes will be run simultaneously. |
| ... | 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. The implementation for the RF- and SPR-supertree are very basic so far and assume that all trees share the same set of taxa.

Value

The function returns an object of class phylo.

Author(s)

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

References

Baum, B. R., (1992) Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, **41**, 3-10.

Ragan, M. A. (1992) Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution*, **1**, 53-58.

See Also

mrp.supertree, [densiTree](#), [RF.dist](#), [SPR.dist](#)

Examples

```
data(Laurasiatherian)
set.seed(1)
bs <- bootstrap.phyDat(Laurasiatherian, FUN = function(x)upgma(dist.hamming(x)), bs=50)

mrp_st <- superTree(bs, rooted=TRUE)
plot(superTree(mrp_st))
## Not run:
rf_st <- superTree(bs, method = "RF")
spr_st <- superTree(bs, method = "SPR")

## End(Not run)
```

| | |
|----------|--------------------------------|
| treedist | <i>Distances between trees</i> |
|----------|--------------------------------|

Description

treedist computes different tree distance methods and RF.dist the Robinson-Foulds or symmetric distance. The Robinson-Foulds distance only depends on the topology of the trees. If edge weights should be considered wRF.dist calculates the weighted RF distance (Robinson & Foulds 1981). and KF.dist calculates the branch score distance (Kuhner & Felsenstein 1994). path.dist computes the path difference metric as described in Steel and Penny 1993). sprdist computes the approximate SPR distance (Oliveira Martins et al. 2008, de Oliveira Martins 2016).

Usage

```
treedist(tree1, tree2, check.labels = TRUE)

sprdist(tree1, tree2)

SPR.dist(tree1, tree2 = NULL)

RF.dist(tree1, tree2 = NULL, normalize = FALSE, check.labels = TRUE,
        rooted = FALSE)

wRF.dist(tree1, tree2 = NULL, normalize = FALSE, check.labels = TRUE,
        rooted = FALSE)

KF.dist(tree1, tree2 = NULL, check.labels = TRUE, rooted = FALSE)

path.dist(tree1, tree2 = NULL, check.labels = TRUE, use.weight = FALSE)
```

Arguments

| | |
|--------------|--|
| tree1 | A phylogenetic tree (class <code>phylo</code>) or vector of trees (an object of class <code>multiPhylo</code>). See details |
| tree2 | A phylogenetic tree. |
| check.labels | compares labels of the trees. |
| normalize | compute normalized RF-distance, see details. |
| rooted | take bipartitions for rooted trees into account, default is unrooting the trees. |
| use.weight | use <code>edge.length</code> argument or just count number of edges on the path (default) |

Details

The Robinson-Foulds distance between two trees T_1 and T_2 with n tips is defined as (following the notation Steel and Penny 1993):

$$d(T_1, T_2) = i(T_1) + i(T_2) - 2v_s(T_1, T_2)$$

where $i(T_1)$ denotes the number of internal edges and $v_s(T_1, T_2)$ denotes the number of internal splits shared by the two trees. The normalized Robinson-Foulds distance is derived by dividing $d(T_1, T_2)$ by the maximal possible distance $i(T_1) + i(T_2)$. If both trees are unrooted and binary this value is $2n - 6$.

Functions like `RF.dist` returns the Robinson-Foulds distance (Robinson and Foulds 1981) between either 2 trees or computes a matrix of all pairwise distances if a `multiPhylo` object is given.

For large number of trees the distance functions can use a lot of memory!

Value

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

| | |
|---------------------------------------|---|
| <code>symmetric.difference</code> | <code>symmetric.difference</code> or Robinson-Foulds distance |
| <code>branch.score.difference</code> | <code>branch.score.difference</code> |
| <code>path.difference</code> | <code>path.difference</code> |
| <code>weighted.path.difference</code> | <code>weighted.path.difference</code> |

Author(s)

Klaus P. Schliep <klaus.schliep@gmail.com>, Leonardo de Oliveira Martins

References

de Oliveira Martins L., Leal E., Kishino H. (2008) *Phylogenetic Detection of Recombination with a Bayesian Prior on the Distance between Trees*. PLoS ONE **3**(7). e2651. doi: 10.1371/journal.pone.0002651

- de Oliveira Martins L., Mallo D., Posada D. (2016) *A Bayesian Supertree Model for Genome-Wide Species Tree Reconstruction*. Syst. Biol. **65**(3): 397–416, doi:10.1093/sysbio/syu082
- Steel M. A. and Penny P. (1993) *Distributions of tree comparison metrics - some new results*, Syst. Biol., **42**(2), 126–141
- Kuhner, M. K. and Felsenstein, J. (1994) *A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates*, Molecular Biology and Evolution, **11**(3), 459–468
- D.F. Robinson and L.R. Foulds (1981) *Comparison of phylogenetic trees*, Mathematical Biosciences, **53**(1), 131–147
- D.F. Robinson and L.R. Foulds (1979) *Comparison of weighted labelled trees*. In Horadam, A. F. and Wallis, W. D. (Eds.), *Combinatorial Mathematics VI: Proceedings of the Sixth Australian Conference on Combinatorial Mathematics*, Armidale, Australia, 119–126

See Also

[dist.topo](#), [nni](#), [superTree](#), [mast](#)

Examples

```
tree1 <- rtree(100, rooted=FALSE)
tree2 <- rSPR(tree1, 3)
RF.dist(tree1, tree2)
treedist(tree1, tree2)
sprdist(tree1, tree2)
trees <- rSPR(tree1, 1:5)
SPR.dist(tree1, trees)
```

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. |
| method | The agglomeration method to be used. This should be (an unambiguous abbreviation 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)
```

writeDist

Writing and reading distances in phylip and nexus format

Description

`readDist`, `writeDist` and `write.nexus.dist` are useful to exchange distance matrices with other phylogenetic programs.

Usage

```
writeDist(x, file = "", format = "phylip", ...)

write.nexus.dist(x, file = "", append = FALSE, upper = FALSE,
  diag = TRUE, digits = getOption("digits"), taxa = !append)

readDist(file)

## S3 method for class 'dist'
unique(x, incomparables, ...)
```

Arguments

| | |
|---------------------|--|
| <code>x</code> | A dist object. |
| <code>file</code> | A file name. |
| <code>format</code> | file format, default is "phylip", only other option so far is "nexus". |
| <code>...</code> | Further arguments passed to or from other methods. |

| | |
|---------------|---|
| append | logical. If TRUE the nexus blocks will be added to a file. |
| upper | logical value indicating whether the upper triangle of the distance matrix should be printed. |
| diag | logical value indicating whether the diagonal of the distance matrix should be printed. |
| digits | passed to format inside of <code>write.nexus.dist</code> . |
| taxa | logical. If TRUE a taxa block is added. |
| incomparables | Not used so far. |

Value

an object of class `dist`

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

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

To compute distance matrices see [dist.ml](#) [dist.dna](#) and [dist.p](#) for pairwise polymorphism p-distances

Examples

```
data(yeast)
dm <- dist.ml(yeast)
writeDist(dm)
write.nexus.dist(dm)
```

yeast

Yeast alignment (Rokas et al.)

Description

Alignment of 106 genes of 8 different species of yeast.

References

Rokas, A., Williams, B. L., King, N., and Carroll, S. B. (2003) Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, **425**(6960): 798–804

Examples

```
data(yeast)
str(yeast)
```

Index

- *Topic **IO**
 - read.aa, [61](#)
- *Topic **\textasciitildekw1**
 - ancestral.pml, [7](#)
- *Topic **\textasciitildekw2**
 - ancestral.pml, [7](#)
 - bab, [11](#)
- *Topic **classif**
 - ldfactorial, [40](#)
 - treedist, [71](#)
- *Topic **cluster**
 - allSplits, [4](#)
 - allTrees, [6](#)
 - bab, [11](#)
 - bootstrap.pml, [12](#)
 - coalSpeciesTree, [16](#)
 - createLabel, [19](#)
 - delta.score, [20](#)
 - designTree, [23](#)
 - discrete.gamma, [25](#)
 - dist.hamming, [26](#)
 - dist.p, [28](#)
 - distanceHadamard, [30](#)
 - fitch, [31](#)
 - getClans, [33](#)
 - getRoot, [35](#)
 - hadamard, [37](#)
 - lento, [41](#)
 - mast, [42](#)
 - maxCladeCred, [43](#)
 - modelTest, [44](#)
 - multiphyDat2pmlPart, [46](#)
 - NJ, [49](#)
 - nni, [50](#)
 - phyDat, [51](#)
 - pml.control, [54](#)
 - pmlCluster, [57](#)
 - pmlMix, [59](#)
 - read.nexus.splits, [62](#)
 - simSeq, [65](#)
 - splitsNetwork, [68](#)
 - superTree, [70](#)
 - upgma, [73](#)
 - writeDist, [74](#)
- *Topic **datasets**
 - chloroplast, [14](#)
 - Laurasiatherian, [39](#)
 - yeast, [75](#)
- *Topic **hplot**
 - consensusNet, [17](#)
 - neighborNet, [47](#)
- *Topic **manip**
 - cophenetic.network, [18](#)
- *Topic **models**
 - SH.test, [64](#)
 - SOWH.test, [67](#)
- *Topic **package**
 - phangorn-package, [3](#)
- *Topic **plot**
 - as.network, [8](#)
 - cladePar, [15](#)
 - densiTree, [21](#)
 - lento, [41](#)
- acctrans(fitch), [31](#)
- ace, [8](#)
- acgt2ry(phyDat), [51](#)
- addConfidences(createLabel), [19](#)
- addTrivialSplits(allSplits), [4](#)
- AIC, [45](#)
- AICc(modelTest), [44](#)
- allCircularSplits(allSplits), [4](#)
- allSitePattern(phyDat), [51](#)
- allSplits, [4](#)
- allTrees, [6](#), [50](#)
- ancestral.pars, [15](#), [32](#)
- ancestral.pars(ancestral.pml), [7](#)
- ancestral.pml, [7](#), [32](#), [57](#)
- anova, [45](#)

- as.AAbin.phyDat (phyDat), 51
- as.bitsplits.splits (allSplits), 4
- as.character.phyDat (phyDat), 51
- as.data.frame.phyDat (phyDat), 51
- as.DNAbin, 53
- as.DNAbin.phyDat (phyDat), 51
- as.igraph, 10
- as.Matrix (allSplits), 4
- as.matrix.splits (allSplits), 4
- as.MultipleAlignment (phyDat), 51
- as.networkx, 5, 8, 20, 64
- as.phyDat (phyDat), 51
- as.phylo, 74
- as.phylo.splits (allSplits), 4
- as.prop.part.splits (allSplits), 4
- as.splits, 20, 41, 64
- as.splits (allSplits), 4

- bab, 11, 15, 32
- baseFreq (phyDat), 51
- bootstrap.phyDat (bootstrap.pml), 12
- bootstrap.pml, 12, 32, 57
- BranchAndBound (bab), 11

- c.phyDat (phyDat), 51
- c.splits (allSplits), 4
- cbind.phyDat (phyDat), 51
- chloroplast, 14
- CI, 15, 32, 35
- cladePar, 15
- coalSpeciesTree, 16
- codon2dna (phyDat), 51
- compatible (allSplits), 4
- consensus, 36, 43
- consensusNet, 10, 14, 17, 43, 48, 69
- cophenetic, 19
- cophenetic.networkx, 18, 48
- cophenetic.splits (cophenetic.networkx), 18
- createLabel, 19

- delta.score, 20
- densiTree, 10, 21, 71
- designSplits (designTree), 23
- designTree, 23, 69
- dfactorial, 12
- dfactorial (ldfactorial), 40
- di2multi, 36
- discrete.gamma, 25
- dist.dna, 27, 29, 49, 75
- dist.hamming, 21, 26, 29, 49, 74
- dist.logDet (dist.hamming), 26
- dist.ml, 75
- dist.ml (dist.hamming), 26
- dist.p, 27, 28, 75
- dist.topo, 73
- distanceHadamard, 5, 10, 18, 24, 30, 37, 38, 48, 69
- distinct.splits (allSplits), 4
- diversity (getClans), 33
- dna2codon (phyDat), 51
- DNAbin, 53

- edQt (discrete.gamma), 25
- evonet, 10

- factorial, 40
- fastme, 24, 49, 74
- fhm (hadamard), 37
- fitch, 15, 31

- genlight2phyDat (phyDat), 51
- getClans, 32, 33
- getClips (getClans), 33
- getDiversity (getClans), 33
- getRoot, 35
- getSlices (getClans), 33

- h2st (hadamard), 37
- h4st (hadamard), 37
- hadamard, 10, 30, 37, 41
- hclust, 73, 74
- howmanytrees, 40

- identify.networkx, 38
- image.phyDat (phyDat), 51

- jitter, 23

- KF.dist (treedist), 71

- Laurasiatherian, 39
- layout_with_kk, 10
- ldfactorial, 40
- lento, 5, 18, 30, 38, 41, 48, 64
- lli (discrete.gamma), 25

- mast, 42, 73
- matchSplits (allSplits), 4

- maxCladeCred, [18](#), [43](#)
- midpoint (getRoot), [35](#)
- modelTest, [44](#), [57](#)
- multiphyDat2pmlPart, [46](#)
- neighborNet, [10](#), [18](#), [30](#), [47](#)
- networkx (as.networkx), [8](#)
- NJ, [32](#), [49](#), [74](#)
- nj, [49](#)
- nni, [6](#), [32](#), [50](#), [73](#)
- nnls.networkx (designTree), [23](#)
- nnls.phylo (designTree), [23](#)
- nnls.splits (designTree), [23](#)
- nnls.tree, [74](#)
- nnls.tree (designTree), [23](#)
- nodelabels, [14](#)
- optim.parsimony (fitch), [31](#)
- optim.pml, [14](#)
- optim.pml (pml.control), [54](#)
- pace (ancestral.pml), [7](#)
- parsimony, [8](#), [15](#), [35](#)
- parsimony (fitch), [31](#)
- path.dist (treedist), [71](#)
- phangorn (phangorn-package), [3](#)
- phangorn-package, [3](#)
- phyDat, [35](#), [51](#), [62](#), [66](#)
- phyDat2alignment (phyDat), [51](#)
- phyDat2MultipleAlignment (phyDat), [51](#)
- plot.networkx, [18](#), [23](#), [30](#), [38](#), [39](#), [48](#), [69](#)
- plot.networkx (as.networkx), [8](#)
- plot.phylo, [14](#), [16](#), [20](#), [23](#), [57](#)
- plotAnc (ancestral.pml), [7](#)
- plotBS (bootstrap.pml), [12](#)
- pml, [8](#), [14](#), [26](#), [32](#), [45](#), [47](#), [59](#), [60](#), [64](#), [66](#), [68](#)
- pml (pml.control), [54](#)
- pml.control, [54](#)
- pml.fit (discrete.gamma), [25](#)
- pml.free (discrete.gamma), [25](#)
- pml.init (discrete.gamma), [25](#)
- pmlCluster, [47](#), [57](#), [60](#), [64](#), [68](#)
- pmlMix, [26](#), [47](#), [53](#), [57](#), [59](#), [59](#)
- pmlPart, [26](#), [57](#), [59](#), [60](#), [64](#), [68](#)
- pmlPart (multiphyDat2pmlPart), [46](#)
- pmlPart2multiPhylo
 (multiphyDat2pmlPart), [46](#)
- pmlPen (pmlMix), [59](#)
- PNJ (NJ), [49](#)
- pratchet, [12](#), [15](#)
- pratchet (fitch), [31](#)
- presenceAbsence (createLabel), [19](#)
- print.splits (allSplits), [4](#)
- prop.part, [5](#), [43](#), [64](#)
- pruneTree (getRoot), [35](#)
- random.addition (fitch), [31](#)
- read.aa, [53](#), [61](#)
- read.alignment, [62](#)
- read.dna, [53](#), [62](#)
- read.GenBank, [62](#)
- read.nexus.data, [53](#)
- read.nexus.networkx (read.nexus.splits),
 [62](#)
- read.nexus.splits, [5](#), [62](#)
- read.phyDat (phyDat), [51](#)
- readDist (writeDist), [74](#)
- removeTrivialSplits (allSplits), [4](#)
- removeUndeterminedSites (phyDat), [51](#)
- RF.dist, [20](#), [71](#)
- RF.dist (treedist), [71](#)
- RI, [32](#), [35](#)
- RI (CI), [15](#)
- rNNI (nni), [50](#)
- root, [8](#), [36](#)
- rSPR (nni), [50](#)
- rtree, [6](#)
- sankoff, [15](#)
- sankoff (fitch), [31](#)
- SH.test, [47](#), [57](#), [59](#), [64](#), [68](#)
- simSeq, [65](#), [68](#)
- SOWH.test, [14](#), [64](#), [66](#), [67](#)
- speciesTree, [17](#)
- splits (allSplits), [4](#)
- splitsNetwork, [10](#), [18](#), [24](#), [48](#), [68](#)
- SPR.dist, [42](#), [50](#), [71](#)
- SPR.dist (treedist), [71](#)
- sprdist (treedist), [71](#)
- subset.phyDat (phyDat), [51](#)
- superTree, [70](#), [73](#)
- treedist, [71](#)
- unique.dist (writeDist), [74](#)
- unique.phyDat (phyDat), [51](#)
- unique.splits (allSplits), [4](#)
- UNJ (NJ), [49](#)

upgma, [24](#), [49](#), [73](#)

wpgma (upgma), [73](#)

wRF.dist (treedist), [71](#)

write.nexus.dist (writeDist), [74](#)

write.nexus.networkx

(read.nexus.splits), [62](#)

write.nexus.splits (read.nexus.splits),
[62](#)

write.phyDat (phyDat), [51](#)

write.splits (read.nexus.splits), [62](#)

writeDist, [27](#), [74](#)

yeast, [75](#)