

# Package ‘phangorn’

September 3, 2010

**Title** Phylogenetic analysis in R

**Version** 1.1-1

**Date** 2010-09-03

**Author** Klaus Schliep

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

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

**Imports** ape, stats

**Suggests** multicore, seqLogo

**Depends** quadprog, ape (>= 2.5.3)

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2010-09-03 19:38:33

## R topics documented:

allTrees . . . . .	2
Ancestors . . . . .	3
ancestral.pml . . . . .	4
as.splits . . . . .	5
bootstrap.pml . . . . .	6
chloroplast . . . . .	7
designTree . . . . .	7
dfactorial . . . . .	8
dist.hamming . . . . .	9
distanceHadamard . . . . .	10
getClans . . . . .	11
hadamard . . . . .	13

Laurasiatherian . . . . .	14
lento . . . . .	15
modelTest . . . . .	16
NJ . . . . .	17
nni . . . . .	18
parsimony . . . . .	19
phyDat . . . . .	20
pml . . . . .	22
pmlCluster . . . . .	24
pmlMix . . . . .	26
pmlPart . . . . .	28
read.aa . . . . .	30
SH.test . . . . .	31
simSeq . . . . .	32
splitsNetwork . . . . .	33
treedist . . . . .	34
upgma . . . . .	35
yeast . . . . .	36
<b>Index</b>	<b>37</b>

---

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>

**Examples**

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

---

Ancestors

*tree utility function*

---

**Description**

Functions for describing relationships among phylogenetic nodes.

**Usage**

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

**Arguments**

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

**Value**

a vector containing the indices of the nodes.

**See Also**

these functions are inspired by `treewalk` in `phylobase` package

**Examples**

```
tree = rtree(10)
plot(tree, show.tip.label = FALSE)
nodelabels()
tiplabels()
Ancestors(tree, 1, "all")
Children(tree, 11)
Descendants(tree, 11, "tips")
Siblings(tree, 3)
```

---

ancestral.pml      *Ancestral character reconstruction.*

---

### Description

Marginal reconstruction of the ancestral character states (of the root).

### Usage

```
ancestral.pml(object, type = "ml", ...)  
ancestral.pars(tree, data, ...)
```

### Arguments

object	an object of class pml
type	either "ml" or "bayes"
tree	a tree, i.e. an object of class pml
data	an object of class phyDat
...	Further arguments passed to or from other methods.

### Value

A matrix containing the

### Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

### References

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

### 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(anc.ml[1:10,]), ic.scale=FALSE)  
seqLogo(t(anc.p[1:10,]), ic.scale=FALSE)  
  
## End(Not run)
```

---

`as.splits`*Splits representation of graphs and trees.*

---

**Description**

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

**Usage**

```
as.splits(x, ...)  
## S3 method for class 'phylo':  
as.splits(x, ...)  
## S3 method for class 'multiPhylo':  
as.splits(x, ...)  
compatible(obj)
```

**Arguments**

<code>x</code>	An object of class <code>phylo</code> or <code>multiPhylo</code> .
<code>...</code>	Further arguments passed to or from other methods.
<code>obj</code>	an object of class <code>splits</code> .

**Value**

`as.splits` returns an object of class `splits`, which is mainly a list of splits. `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](#), [distanceHadamard](#)

**Examples**

```
as.splits(rtree(5))
```

bootstrap.pml      *Bootstrap*

---

### Description

`bootstrap.pml` performs (non-parametric) bootstrap analysis and `bootstrap.phyDat` produces a list of bootstrapped data sets. Under Linux the bootstrap is performed in parallel, if the multicore package is loaded. `plotBS` plots a phylogenetic tree with the with the bootstrap values assigned to the (internal) edges.

### Usage

```
bootstrap.pml(x, bs = 100, trees = TRUE, ...)  
bootstrap.phyDat(x, FUN, bs = 100, ...)  
plotBS(tree, BStrees, type="unrooted", bs.col="black", bs.adj=c(0.5, 0.5), ...)
```

### Arguments

<code>x</code>	an object of class <code>pml</code> or <code>phyDat</code> .
<code>bs</code>	number of bootstrap samples.
<code>trees</code>	return trees only (default) or whole <code>pml</code> objects.
<code>...</code>	further parameters used by <code>optim.pml</code> or <code>plot.phylo</code> .
<code>FUN</code>	the function to estimate the trees.
<code>tree</code>	The tree on which edges the bootstrap values are plotted.
<code>BStrees</code>	a list of trees (object of class "multiPhylo").
<code>type</code>	the type of tree.
<code>bs.col</code>	color of bootstrap support labels.
<code>bs.adj</code>	one or two numeric values specifying the horizontal and vertical justification of the bootstrap labels.

### Value

returns an object of class `multi.phylo` or a list where each element is an object of class `pml`.

### Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

### See Also

[optim.pml](#), [pml](#), [plot.phylo](#)

**Examples**

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

set.seed(1)
bs <- bootstrap.pml(fit, bs=100, optNni=TRUE)
plotBS(fit$tree,bs)

## End(Not run)
```

---

chloroplast

*Chloroplast alignment*

---

**Description**

Amino acid alignment of 15 genes of 19 different chloroplast.

**Usage**

```
data(yeast)
```

**Examples**

```
data(chloroplast)
chloroplast
```

---

designTree

*Compute a design matrix*

---

**Description**

`designTree` and `designSplits` compute design matrices for the estimation of edge length of (phylogenetic) trees using linear models. `designTree` also computes a contrast matrix if the method is "rooted".

**Usage**

```
designTree(tree, method = "unrooted", ...)
designSplits(x, splits = "all", ...)
```

**Arguments**

tree	an object of class <code>phylo</code>
method	design matrix for an "unrooted" or "rooted" ultrametric tree
x	number of taxa.
splits	one of "all", "star"
...	further arguments, passed to other methods.

**Value**

a matrix.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**See Also**

[fastme](#), [distanceHadamard](#)

**Examples**

```
example(NJ)
dm <- as.matrix(dm)
y <- dm[lower.tri(dm)]
X <- designTree(tree)
summary(lm(y~X-1))
```

---

dfactorial

*Arithmetic Operators*

---

**Description**

double factorial function

**Usage**

```
dfactorial(x)
ldfactorial(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](#)

**Examples**

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

---

dist.hamming

*Pairwise Distances from Sequences*

---

**Description**

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

**Usage**

```
dist.hamming(x, ratio = TRUE)  
dist.logDet(x)  
dist.ml(x, model="JC69", ...)
```

**Arguments**

x	an object of class dist.logDet
ratio	compute uncorrected ('p') distance or character difference.
model	One of "JC69", "WAG", "JTT", "LG" or "Dayhoff"
...	Further arguments passed to or from other methods.

**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.

**See Also**

For more distance methods for nucleotide data see [dist.dna](#)

**Examples**

```
data(Laurasiatherian)
dm1 <- dist.hamming(Laurasiatherian)
tree1 <- NJ(dm1)
dm2 <- dist.logDet(Laurasiatherian)
tree2 <- NJ(dm2)
treedist(tree1, tree2)
```

---

distanceHadamard     *Distance Hadamard*

---

**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](#)

**Examples**

```
data(yeast)
dm = dist.hamming(yeast)
dm = as.matrix(dm)
fit = distanceHadamard(dm)
lento(fit)
```

---

getClans

*Clans, slices and clips*


---

**Description**

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

**Usage**

```
getClans(tree)
getClips(tree, all=TRUE)
getSlices(tree)
getDiversity(tree, x, norm=TRUE, var.names = NULL)
```

**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.

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

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.

### 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 data.frame of the equitability index or Shannon divergence and parsimony scores (p-score) for all trees and variables.

### 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

### 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

getDiversity(trees, x, var.names=colnames(X))

```

---

hadamard

*Hadamard Matrices and Fast Hadamard Multiplication*


---

## Description

A collection of functions to perform Hadamard conjugation.

## Usage

```

hadamard(x)
fhm(v)
h2st(obj, eps=0.001)
h4st(obj, levels = c("a", "c", "g", "t"))
write.nexus.splits(obj, file="", weights=NULL)

```

## 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>eps</code>	Threshold value for splits.
<code>levels</code>	levels of the sequences.
<code>file</code>	a file name.
<code>weights</code>	Edge weights.

## Details

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

**Value**

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

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**

- 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](#)

**Examples**

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

data(yeast)
dat = as.character(yeast)
# RY-coding
dat[dat=="a" | dat=="g"] = "r"
dat[dat=="c" | dat=="t"] = "y"
dat = phyDat(dat, type="USER", levels=c("r","y"))
fit = h2st(dat)
# write.nexus.splits(fit, file = "test.nxs")
# read this file into Spectronet or Splitstree to show the network
```

---

Laurasiatherian      *Laurasiatherian data (AWCMEE)*

---

**Description**

Laurasiatherian RNA sequence data

**Usage**

```
data(Laurasiatherian)
```

**Source**

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

**Examples**

```
data(Laurasiatherian)
str(Laurasiatherian)
```

---

lento	<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,
```

**Arguments**

obj	an object of class phylo or splits
xlim	graphical parameter
ylim	graphical parameter
main	graphical parameter
sub	graphical parameter
xlab	graphical parameter
ylab	graphical parameter
bipart	plot bipartition information.
...	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)
```

---

modelTest

*ModelTest*

---

## Description

Comparison of different substitution models

## Usage

```
modelTest(tree, data, model = c("JC", "F81", "K80", "HKY", "SYM", "GTR"), G = TRUE,
```

## Arguments

tree	a phylogenetic tree.
data	an object of class phyDat
model	substitution models to compare with each other
G	(discrete) Gamma model
I	invariant sites
k	rate classes
control	A list of parameters for controlling the fitting process.

## Value

a data.frame containing the AIC and BIC for the tested models.

## Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

## References

Posada, D. (2008) jModelTest: Phylogenetic Model Averaging. *Molecular Biology and Evolution* **25**: 1253-1256

## See Also

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

## Examples

```
## Not run:  
example(NJ)  
modelTest(tree, Laurasiatherian)  
  
## End(Not run)
```

---

NJ

*Neighbor-Joining*

---

## 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.logDet(Laurasiatherian)
tree <- NJ(dm)
plot(tree)
```

---

nni

*Tree rearrangements.*

---

**Description**

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

**Usage**

```
nni(tree)
rSPR(tree, moves=1, n=1)
rNNI(tree, moves=1, n=1)
```

**Arguments**

tree	A phylogenetic tree, object of class phylo.
moves	Number of tree rearrangements to be transformed on a tree.
n	Number of trees to be simulated.

**Value**

an object of class multiPhylo.

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**Examples**

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

---

parsimony	<i>Parsimony tree.</i>
-----------	------------------------

---

### Description

`parsimony` returns the parsimony score of a tree. `optim.parsimony` tries to find the maximum parsimony tree using Nearest Neighbor Interchange (NNI) rearrangements. `pace` returns a (logical) matrix of the ancestral states of the root node. `CI` and `RI` computes the consistency and retention index.

### Usage

```
parsimony(tree, data, method="fitch", ...)  
optim.parsimony(tree, data, method="fitch", cost=NULL, trace=1, ...)  
fitch(tree, data, site = "pscore")  
sankoff(tree, data, cost = NULL, site = "pscore")  
pace(tree, data, ...)  
CI(tree, data)  
RI(tree, data)
```

### Arguments

<code>data</code>	A object of class <code>phyDat</code> containing (dna) sequences.
<code>tree</code>	tree to start the nni search from.
<code>method</code>	one of 'fitch' or 'sankoff'.
<code>cost</code>	A cost matrix for the transitions between two states.
<code>site</code>	return either 'pscore' or 'site' wise parsimony scores.
<code>trace</code>	defines how much information is printed during optimisation.
<code>...</code>	Further arguments passed to or from other methods.

### Value

`parsimony` returns the maximum parsimony score (`pscore`). `optim.parsimony` returns a tree after NNI rearrangements.

### Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

### References

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

### See Also

[ancestral.pml](#), [nni](#), [NJ](#), [pml](#), [getClans](#)

**Examples**

```
data(Laurasiatherian)
dm = dist.logDet(Laurasiatherian)
tree = NJ(dm)
parsimony(tree, Laurasiatherian)
ptree <- optim.parsimony(tree, Laurasiatherian)
```

---

phyDat

*Conversion among Sequence Formats*


---

**Description**

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

**Usage**

```
phyDat(data, type = "DNA", levels = NULL, return.index=TRUE, ...)
read.phyDat(file, format="phylip", type="DNA", ...)
write.phyDat(x, file, format="phylip",...)
## S3 method for class 'DNABin':
as.phyDat(x, ...)
## S3 method for class 'phyDat':
as.character(x, allLevels = TRUE, ...)
## S3 method for class 'phyDat':
as.data.frame(x, ...)
## S3 method for class 'phyDat':
as.DNABin(x, ...)
## S3 method for class 'phyDat':
subset(x, subset, select, ...)
allSitePattern(n, levels=c("a","c","g","t"), names=NULL)
```

**Arguments**

<code>data</code>	An object containing sequences.
<code>x</code>	An object containing sequences.
<code>type</code>	Type of sequences ("DNA", "AA" or "USER").
<code>levels</code>	Level attributes.
<code>return.index</code>	If TRUE returns a index of the site patterns.
<code>file</code>	A file name.
<code>format</code>	File format of the sequence alignment (see details).
<code>n</code>	Number of sequences.
<code>names</code>	Names of sequences.
<code>subset</code>	a subset of taxa.

<code>select</code>	a subset of characters.
<code>allLevels</code>	return original data.
<code>...</code>	further arguments passed to or from other methods.

## Details

If type "USER" a vector has to be give to `levels`. For example `c("a", "c", "g", "t", "-")` would create a data object that can be used in phylogenetic analysis with gaps as fifth state. `allSitePattern` returns all possible site patterns and can be useful in simulation studies. For further details see the vignette `phangorn-specials`. `write.phyDat` calls the function `write.dna` or `write.nexus.data` and `read.phyDat` calls the function `read.dna`, `read.aa` or `read.nexus.data` see for more details over there.

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

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

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

## Value

The functions return an object of class `phyDat`.

## Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

## See Also

[DNABin](#), [as.DNABin](#), [read.dna](#), [read.aa](#) and [read.nexus.data](#) and the example of [pmlMix](#) for the use of `allSitePattern`

## Examples

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

pml

*Likelihood of a tree.***Description**

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

**Usage**

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

**Arguments**

tree	A phylogenetic tree, object of class phylo.
data	The (DNA) alignment.
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	Amino acid models: one of "WAG", "JTT", "Dayhoff" or "LG" or nucleotide model
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.
subs	A (integer) vector same length as Q to specify the optimization of Q
...	Further arguments passed to or from other methods.
epsilon	Stop criterion for optimisation (see details).
maxit	Maximum number of iterations (see details).
trace	Show output during optimization (see details).

## Details

The topology search uses a nearest neighbor interchange (NNI) and the implementation is similar to phyML. The option `model` in `pml` is only used for amino acid models. The option `model` defines the nucleotide model which is getting 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 an 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. For how these models are specified see Posada (2008).

So far 4 amino acid models are supported ("WAG", "JTT", "Dayhoff" and "LG").

If the option `'getRooted'` is set to `TRUE` than the edge lengths of rooted tree are optimized. The tree has to be rooted and ultrametric! No tree rearrangements are yet supported. If `'getRooted=FALSE'` any rooted tree is getting unrooted.

`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 heuristic which works reasonably for small and large trees or alignments. If `trace` is set to zero than no output is shown, if functions are called internally than the trace is decreased by one.

## Value

Returns a list of class `ll.phylo`

<code>logLik</code>	Log likelihood of the tree.
<code>siteLik</code>	Site log likelihoods.
<code>root</code>	likelihood in the root node.
<code>weight</code>	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.

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

Posada, D. (2008) jModelTest: Phylogenetic Model Averaging. *Molecular Biology and Evolution* **25**: 1253-1256

## 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, optNni=TRUE, optGamma=TRUE, optInv=TRUE, optBf=TRUE, optQ=TRUE)

## End(Not run)

# 2-state data (RY-coded)

dat <- as.character(Laurasiatherian)
# RY-coding
dat[dat=="a"] <- "r"
dat[dat=="g"] <- "r"
dat[dat=="c"] <- "y"
dat[dat=="t"] <- "y"
dat <- phyDat(dat, levels=c("r","y"))
fit2ST <- pml(tree, dat, k=4, inv=.25)
fit2ST <- optim.pml(fit2ST,optNni=TRUE, optGamma=TRUE, optInv=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=4, part=NULL, control=pml.control(eps=1e-8, maxi
```

**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.
control	A list of parameters for controlling the fitting process.
...	Further arguments passed to or from other methods.

**Details**

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

**Value**

`pmlCluster` returns a list with elements

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

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**See Also**

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

**Examples**

```
## 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=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(eps=1e-08, maxit=2
```

**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  $\text{edge} + \text{bf} + \text{Q} \sim \text{rate} + \text{shape} + \dots$ , 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**

```
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))

## Not run:
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
#

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)

```

---

pmlPart

*Partition model.*


---

## Description

Model to estimate phylogenies for partitioned data.

## Usage

```
pmlPart(formula, object, control = pml.control(eps=1e-8, maxit=10, trace=1),...)
```

## Arguments

formula	a formula object (see details).
object	an object of class pml or a list of objects of class pml .
control	A list of parameters for controlling the fitting process.
...	Further arguments passed to or from other methods.

## Details

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

"edge", "rate". Each parameters can be used only once in the formula. "rate" and "nni" are only available for the right side of the formula.

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

## 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

sp2 <- pmlPart(~ edge + inv, fits, weight=weight)
sp2
AIC(sp2)

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

---

`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**

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

Anonymous. FASTA format description. <http://www.ncbi.nlm.nih.gov/BLAST/fasta.html>

Felsenstein, J. (1993) Phylip (Phylogeny Inference Package) version 3.5c. Department of Genetics, University of Washington. <http://evolution.genetics.washington.edu/phylip/phylip.html>

**See Also**

[read.dna](#), [read.GenBank](#), [phyDat](#), [read.alignment](#)

---

`SH.test`*Shimodaira-Hasegawa Test*

---

**Description**

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

**Usage**

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

**Arguments**

`...` either a series of objects of class "pml" separated by commas, a list containing such objects or an object of class "pmlPart".

`B` the number of bootstrap replicates.

`data` an object of class "phyDat".

**Value**

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

**Author(s)**

Klaus Schliep <klaus.schliep@gmail.com>

**References**

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](#)

**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)
## 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(tree, l=1000, Q=NULL, bf=NULL, rootseq=NULL, type="DNA",
        model="", levels=NULL, rate=1, ancestral=FALSE)
```

### Arguments

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

### Details

`simSeq` simulates sequence alignments. So far rate variation is not yet implemented, but one can combine different alignments having their own rate. In fact it is possible to generate DNA, RNA, amino acid, or 0/1.

### Value

`simSeq` returns an object of class `phyDat`.

### Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

**See Also**[phyDat](#)**Examples**

```
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 = phangorn::discrete.gamma(1,4)
data1 = simSeq(tree, l = 100, type="AA", model="WAG", rates[1])
data2 = simSeq(tree, l = 100, type="AA", model="WAG", rates[2])
data3 = simSeq(tree, l = 100, type="AA", model="WAG", rates[3])
data4 = simSeq(tree, l = 100, type="AA", model="WAG", rates[4])
data <- c(data1,data2, data3, data4)

write.phyDat(data, file="temp.dat", format="sequential",nbc0l = -1, colsep = "")
unlink("temp.dat")
```

---

splitsNetwork

*Phylogenetic Network*

---

**Description**

splitsNetwork estimates a splits graph from a distance matrix.

**Usage**

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

**Arguments**

dm	A distance matrix.
gamma	penalty value for the L1 constraint.
lambda	penalty value for the L2 constraint.
weight	a vector of weights.

**Details**

`splitsNetwork` fits phylogenetic networks using L1, L2 and non-negativity 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 constraints.

**Value**

`splitsNetwork` returns a matrix. 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**

K. P. Schliep (2009). Some Applications of statistical phylogenetics (PhD Thesis)

**See Also**

[distanceHadamard](#), [designTree](#)

**Examples**

```
data(yeast)
dm = dist.ml(yeast)
fit = splitsNetwork(dm)
write.nexus.splits(fit)
```

---

treedist

*Distances between trees*

---

**Description**

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

**Usage**

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

**Arguments**

tree1            A phylogenetic tree.  
 tree2            A phylogenetic tree.  
 check.labels    compares labels of the trees.

**Value**

treedist returns a vector containing the following tree distance methods

symmetric.difference  
                          symmetric.difference or Robinson-Foulds distance

branch.score.difference  
                          branch.score.difference

path.difference  
                          path.difference

weighted.path.difference  
                          weighted.path.difference

**Author(s)**

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

**References**

Steel M. A. and Penny P. (1993) *Distributions of tree comparison metrics - some new results*, Syst. Biol.,42(2), 126-141

---

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](#), [NJ](#), [as.phylo](#), [fastme](#)

**Examples**

```
data(Laurasiatherian)
dm = dist.logDet(Laurasiatherian)
tree = upgma(dm)
plot(tree)
```

---

yeast

*Yeast alignment (Rokas et al.)*

---

**Description**

Alignment of 106 genes of 8 different species of yeast.

**Usage**

```
data(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, 29

## \*Topic **\textasciitildekwd1**

ancestral.pml, 3

## \*Topic **\textasciitildekwd2**

ancestral.pml, 3

## \*Topic **classif**

dfactorial, 7

treedist, 33

## \*Topic **cluster**

allTrees, 1

as.splits, 4

bootstrap.pml, 5

designTree, 6

dist.hamming, 8

distanceHadamard, 9

getClans, 10

hadamard, 12

lento, 14

modelTest, 15

NJ, 16

nni, 17

parsimony, 18

phyDat, 19

pml, 21

pmlCluster, 23

pmlMix, 25

pmlPart, 27

simSeq, 31

splitsNetwork, 32

upgma, 34

## \*Topic **datasets**

chloroplast, 6

Laurasiatherian, 13

yeast, 35

## \*Topic **misc**

Ancestors, 2

## \*Topic **models**

SH.test, 30

## \*Topic **plot**

lento, 14

acgt2ry (*phyDat*), 19

allSitePattern (*phyDat*), 19

allTrees, 1

Ancestors, 2

ancestral.pars (*ancestral.pml*), 3

ancestral.pml, 3, 18

anova, 16

as.character.phyDat (*phyDat*), 19

as.data.frame.phyDat (*phyDat*), 19

as.DNAbin, 20

as.DNAbin.phyDat (*phyDat*), 19

as.phyDat (*phyDat*), 19

as.phylo, 35

as.splits, 4, 15

bootstrap.phyDat (*bootstrap.pml*),  
5

bootstrap.pml, 5

Children (*Ancestors*), 2

chloroplast, 6

CI, 11

CI (*parsimony*), 18

compatible (*as.splits*), 4

Descendants (*Ancestors*), 2

designSplits (*designTree*), 6

designTree, 6, 33

dfactorial, 7

dist.dna, 9, 17

dist.hamming, 8, 17

dist.logDet (*dist.hamming*), 8

dist.ml (*dist.hamming*), 8

distanceHadamard, 5, 7, 9, 12, 13, 33

DNAbin, 20

factorial, 8

fastme, 7, 17, 35

fhm (*hadamard*), 12  
fitch (*parsimony*), 18

getClans, 10, 18  
getClips (*getClans*), 10  
getDiversity (*getClans*), 10  
getSlices (*getClans*), 10

h2st (*hadamard*), 12  
h4st (*hadamard*), 12  
hadamard, 9, 12, 15  
hclust, 34, 35

Laurasiatherian, 13  
ldfactorial (*dfactorial*), 7  
lento, 5, 9, 13, 14

modelTest, 15

NJ, 16, 18, 35  
nj, 17  
nni, 17, 18

optim.parsimony (*parsimony*), 18  
optim.pml, 6  
optim.pml (*pml*), 21

pace (*parsimony*), 18  
parsimony, 11, 18  
phyDat, 11, 19, 29, 32  
plot.phylo, 6  
plotBS (*bootstrap.pml*), 5  
pml, 6, 16, 18, 21, 24, 26, 28, 30  
pmlCluster, 23, 26, 28, 30  
pmlMix, 20, 24, 25, 28  
pmlPart, 24, 26, 27, 30  
pmlPen (*pmlMix*), 25  
PNJ (*parsimony*), 18  
prop.part, 5

read.aa, 20, 29  
read.alignment, 29  
read.dna, 20, 29  
read.GenBank, 29  
read.nexus.data, 20  
read.phyDat (*phyDat*), 19  
RF.dist (*treedist*), 33  
RI, 11  
RI (*parsimony*), 18  
rNNI (*nni*), 17  
rSPR (*nni*), 17

sankoff (*parsimony*), 18  
SH.test, 24, 28, 30  
Siblings (*Ancestors*), 2  
simSeq, 31  
splitsNetwork, 32  
subset.phyDat (*phyDat*), 19

treedist, 33

UNJ (*NJ*), 16  
upgma, 17, 34

wpgma (*upgma*), 34  
write.nexus.splits (*hadamard*), 12  
write.phyDat (*phyDat*), 19

yeast, 35