Calculation of evolutionary rates and rate shifts with the RRphylo method

Contents

1. **RRphylo** function for rates estimation

   1.1. Technical details about RRphylo penalization function

2. **search.shift** function for rate shift location
3. **setBM** function for producing phenotypes with desired trend in rates or mean value over time
4. **sizedsubtree** function to randomly extract subclades of a given size from a tree
5. **evo.dir** function to quantify direction, size and rate of evolutionary change in multivariate traits along node-to-tip paths and between species.
6. **retrieve.angle** function to extract a user-specified subset of the *evo.dir* results
7. **angle.matrix** function to compute ontogenetic vectors for each species and compare them between species pairs.
8. **makeL** function to produce a matrix including the branch lengths along a root-to-tip path for the whole phylogeny.
9. **makeL1** function to produce a matrix including the branch lengths along a root-to-node path for the whole phylogeny.
10. **getMommy** function to derive the node path from a given node or species to the root of the phylogeny.
11. **makeFossil** function to randomly change the lengths of the leaves.
12. **swap.phylo** function to test the effect of phylogenetic uncertainty on rate shifts found at a particular node
13. **swapONE** function to create alternative phylogenies from a given tree

14. **plotRates** function to plot RRphylo rates at a specified node
15. **search.trend** function to search for evolutionary trends in phenotypes and rates
16. **getSis** function to get sister clade
17. **distNodes** function to find distance between nodes and tips
18. **search.conv** function to search for morphological convergence among species and clades
19. **PGLS_fossil** function to perform Phylogenetic Generalized Least Square with fossil phylogenies
20. **StableTraitsR** function to run StableTraits from within R
21. **overftiRR** function for testing RRphylo methods overfit
22. **scaleTree** function for phylogenetic tree calibration
23. **phyloclust** function to test for phylogenetic clustering

Depends ape, phytools, geiger, stats4, foreach, doParallel, pvclust, mvMORPH, lmtest, parallel, phangorn, rlist, scales, R.utils, binr, RColorBrewer, nlme, penalized, smatr, data.tree, car, outliers, emmeans, picante, vegan, plotrix, cluster, tseries, ddpcr, geomorph

1. `RRphylo`

Title *Evolutionary rates computation along phylogenies*

Description

The function *RRphylo* performs the phylogenetic ridge regression. It takes a tree and a vector of tip data (phenotypes) as entries, calculates the penalization factor λ, produces the matrices of tip to root, and node to root distances (matrix **L** and **L′**), the vector of ancestral state

estimates, the vector of predicted phenotypes, and the rates vector $\hat{\beta}$ for all the branches of the tree. For multivariate data, rates are given as both one entry per variable, and as a multivariate vector obtained by computing the Euclidean Norm of the rate entries per observation.

Usage

```
RRphylo(tree,y,cov=NULL,rootV=NULL,aces=NULL,x1=NULL,aces.x1=NULL,
clus=0.5)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric or fully dichotomous

y either a single vector variable or a multivariate dataset of class 'matrix'

cov the covariate to be indicated if its effect on the rates must be accounted for. In this case residuals of the covariate versus the rates are used as rates. 'cov' must be as long as the number of nodes plus the number of tips of the tree, which can be obtained by running *RRphylo* on the covariate as well, and taking the vector of ancestral states and tip values to form the covariate, as in the example below.

rootV phenotypic value (values if multivariate) at the tree root. If 'rootV'=NULL the function "learns" about the root value from the 10% tips being closest in time to the tree root, weighted by their temporal distance from the root itself (close tips phenotypes weigh more than more distant tips).

aces a named vector (or matrix if 'y' is multivariate) of ancestral character values at nodes. Names correspond to the nodes in the tree.

x1 the additional predictor to be indicated to perform the multiple version of *RRphylo*. 'x1' vector must be as long as the number of nodes plus the number of tips of the tree,

which can be obtained by running *RRphylo* on the predictor as well, and taking the vector of ancestral states and tip values to form the 'x1'.

aces.x1 a named vector of ancestral character values at nodes for 'x1'. It must be indicated if both 'aces' and 'x1' are specified. Names correspond to the nodes in the tree.

clus the proportion of clusters to be used in parallel computing (only if 'y' is multivariate).

Details

Details about *RRphylo* are in paragraph 1.1.

Value

tree the tree used by *RRphylo*. The fully dichotomous version of the tree argument. For trees with polytomies, the tree is resolved by using *multi2di* function in the package ape. If the latter is a dichotomous tree, the two trees will be identical

tip.path a *n x m* matrix, where n=number of tips and m=number of branches (i.e. n*2-1). Each row represents the branch lengths along a root-to-tip path.

node.path a *n x n* matrix, where n=number of internal branches. Each row represents the branch lengths along a root-to-node path.

rates single rate values computed for each branch of the tree. If 'y' is a single vector variable, rates are equal to $multiple.rates. If 'y' is a multivariate dataset, rates are computed as the square root of the sum of squares of each row of $multiple.rates.

aces the phenotypes reconstructed at nodes.

predicted.phenotypes the vector of estimated tip values. It is a matrix in the case of multivariate data.

multiple.rates a *n x m* matrix, where n= number of branches (i.e. n*2-1) and m = number of variables. For each branch, the column entries represent the evolutionary rate.

lambda the penalization factor fitted within *RRphylo* by the inner function *optL*. With multivariate data, several *optL* runs are performed. Hence, the function provides a single 'lambda' for each individual variable.

ace.values if aces are specified, the function returns a data frame containing the corresponding node number on the *RRphylo* tree for each node, along with estimated values.

x1.rate if 'x1' is specified, the function returns the partial regression coefficient for 'x1'.

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
### DataOrnithodirans$massdino->massdino

## Case 1. "RRphylo" without accounting for the effect of a
## covariate
# RRphylo(tree=treedino,y=massdino)

## Case 2. "RRphylo" accounting for the effect of a covariate
## "RRphylo" on the covariate in order to retrieve ancestral state
## values
# RRphylo(tree=treedino,y=massdino)->RRcova
```

```
# c(RRcova$aces,massdino)->cov.values
# c(rownames(RRcova$aces),names(massdino))->names(cov.values)

# RRphylo(tree=treedino,y=massdino,cov=cov.values)

## Case 3. "RRphylo" specifying the ancestral states
### data("DataCetaceans")
### DataCetaceans$treecet->treecet
### DataCetaceans$masscet->masscet
### DataCetaceans$brainmasscet->brainmasscet
### DataCetaceans$aceMyst->aceMyst

# RRphylo(tree=treecet,y=masscet,aces=aceMyst)

## Case 4. Multiple "RRphylo"
# drop.tip(treecet,treecet$tip.label[match(names(brainmasscet),
# treecet$tip.label)])->treecet.multi
# masscet[match(treecet.multi$tip.label,
# names(masscet))]->masscet.multi

# RRphylo(tree=treecet.multi, y=masscet.multi)->RRmass.multi

# RRmass.multi$aces[,1]->acemass.multi
# c(acemass.multi,masscet.multi)->x1.mass
```

```
# RRphylo(tree=treecet.multi,y=brainmasscet,x1=x1.mass)
```

1.1. Phylogenetic Ridge Regression penalization within RRphylo

Under phylogenetic Ridge Race regression, the vector of rates at individual branches 'betas' is calculated as:

```
betas <- (solve(t(L) %*% L + lambda * diag(ncol(L))) %*%
      t(L)) %*% as.matrix(y)
```

and the vector of predicted phenotypes as:

```
y <- L %*% betas
```

Where **L** is the matrix of root to tip distances provided by *RRphylo*, and 'lambda' is the penalization factor fitted within *RRphylo*. Note that without penalization, the vector of rates is calculated as to compute the phenotypic vector perfectly:

```
L%*%t(L)%*%solve(L%*%t(L))%*%y
```

Kratch and McHardy (2014) used L2 (quadratic) penalization to find the value of the parameter λ. Within *RRphylo*, λ is fitted by an inner function *optL*, which is written as to minimize the rate variation along individual branches of the tree, thereby acting conservatively in terms of the chance to find rate shifts. We illustrate here the effect of assuming different λ values in *RRphylo*. We produced a random, non-ultrametric tree by using *rtree* function in the R package ape. Secondly, we simulated a phenotype evolving on such tree under the Brownian motion model (BM) of evolution. Then, we produced a uniform distribution of λ values, ranging from $1*10^{-5}$ to 10, and computed the phenotypes predicted by *RRphylo* regression under the values of the uniform distribution of lambdas. Eventually, we calculated the resulting phylogenetic signal K (Blomberg et al. 2003) for each of the simulated phenotype. The plot of Ks versus the corresponding λ values is reported in figure 1.

```
# ## produce the tree and phenotype
# library(ape)
# library(lmtest)
# rtree(100)->tree
# fastBM(tree)->y
#
## perform RRphylo
# RRphylo(tree,y)->RR
# RR$tip.path->L
# RR$node.path->L1
#
## set the value at the tree root
```

```
# u <- data.frame(y, (1/diag(vcv(tree))^2))
# u <- u[order(u[, 2], decreasing = TRUE), ]
# u1 <- u[1:(dim(u)[1] * 0.1), ]
# rootV <- weighted.mean(u1[, 1], u1[, 2])
#
## compute the phylogenetic signal at different values of lambda, taken
## from a uniform distribution of lambda values
# seq(0.00001,10, length.out=10000)->lambdaD
# lambdaD[-5001]->lambdaD
# f=function(lambda){
# require(phytools)
#   betas <- (solve(t(L) %*% L + lambda * diag(ncol(L))) %*%
#               t(L)) %*% (as.matrix(y) - rootV)
#   aceRR <- (L1 %*% betas[1:Nnode(tree), ]) + rootV
#   y.hat <- (L %*% betas) + rootV
#   return(phylosig(tree,y.hat,test=TRUE))
# }
# library(parallel)
# library(pbapply)
# cl<-makeCluster(detectCores()*0.5)
# clusterExport(cl, c("f", "L", "L1", "y", "tree","rootV"))
# pbsapply(lambdaD,f,cl=cl)->Ks
# stopCluster(cl)
# unlist(Ks[1,])->KS
# unlist(Ks[2,])->Ps
#
#
```

```
## plot the phylogenetic signals versus lambdas. Eventually, the real
## (fitted) lambda is superimposed on the plot
# library(phytools)
# col<-rep("black",length(KS))
# col[which(KS>phylosig(tree,y)*0.75 & KS<phylosig(tree,y)*1.25)]<-"red"
# plot(lambdaD,KS,col=col,xlab="fixed lambda value",ylab="fitted
phylogenetic signal K",cex=1.5,cex.lab=1.8)
# abline(v=RR$lam,col="purple",lwd=2.5)
```

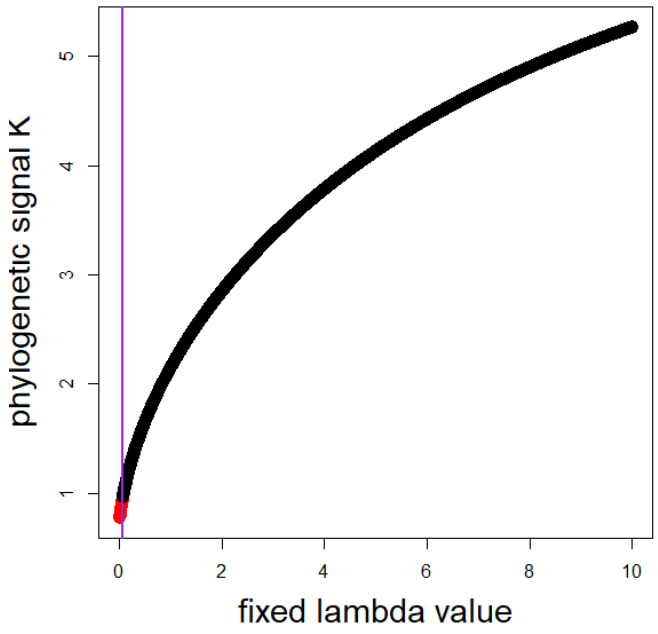

Figure 1 The phylogenetic signal K at different values of the penalization factor λ. The initial phenotype was simulated according to the Brownian motion model of evolution. As expected, the fitted λ value (indicated by the purple vertical line) produces a phylogenetic signal close to 1.

From the simulations it is empirically possible to note that the Brownian motion model of evolution (which is expected to produce values of phylogentic signal K close to 1) corresponds to small absolute values of λ . It is important to notice that when the value of λ is fitted to the original tree and phenotype (rather than taken from a uniform distribution of λ values as in the simulations), *RRphylo* produces phenotypes evolving according to BM, as expected (the fitted λ value is represented by the purple vertical line in figure 1), which implies the function works properly.

However, values of K consistently close to 1 appear at very low values of λ values. To better appreciate the effect of assuming different λ values, we plotted the original tree with the branches colored according to the phenotypes simulated under BM by using the function *contMap* in the R package phytools (Figure 2, center), produced with the fitted λ (Figure 2, right), and with a λ values producing a phylogenetic signal close to 1 (see the red dots in the negative realm of λ in Figure 1). Under these small values of λ, the phenotypes deviate significantly from the BM expectation (Figure 2, left). This means that *RRphylo* produces, beyond a legitimate value of K, a distribution of phenotypes at the tips very close to the original distribution.

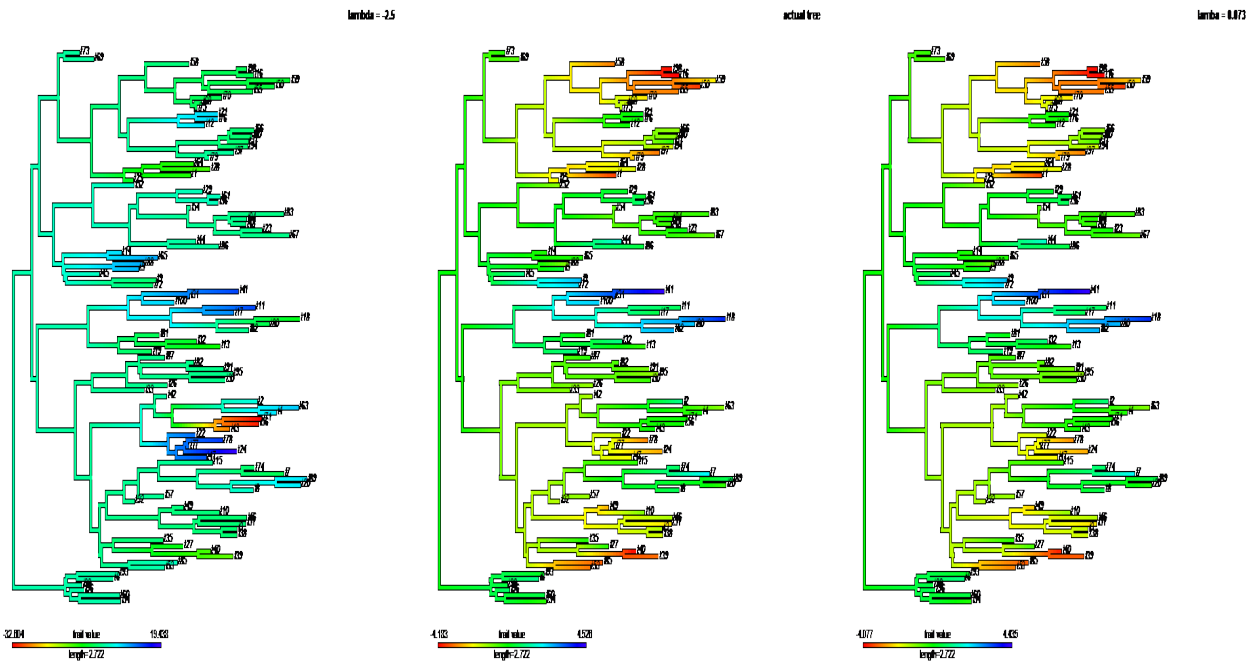

Figure 2 The distribution of phenotypes produced with λ values giving a K value close to the original, Brownian motion simulation. The original phenotypes are reported in the tree at the center of the figure. The phenotypes with the λ value fitted in *RRphylo* are to the right. The tree to the left represents the phenotypes produced with a small value of λ that still gives a K value close to 1.

We next calculated the likelihood surface for the uniform distribution of λ values, by using the package bbmle (Bolker 2017). Of course, the likelihood peaks (in terms of deviance) at the fitted value of λ, where deviance is the difference in negative log-likelihood from the likelihood at the MLE λ. The y-axis in Figure 3 represents the signed square root of the deviance difference (z), while the x-axis is the uniform distribution of starting λ values as used in the simulations. Please notice that the right end of the confidence interval is rather flat below the alpha level (95%). This could be adjusted by changing the standard error manually. Please refer to the bbmle2 package vignette for full explanation.

```
t<-tree
library(bbmle)
optL <- function(lambda) {
  y <- scale(y)
  betas <- (solve(t(L) %*% L + lambda * diag(ncol(L))) %*%
              t(L)) %*% (as.matrix(y) - rootV)
  aceRR <- (L1 %*% betas[1:Nnode(t), ]) + rootV
  y.hat <- (L %*% betas) + rootV
  Rvar <- array()
  for (i in 1:Ntip(t)) {
    ace.tip <- betas[match(names(which(L[i, ] != 0)),
                           rownames(betas)), ]
    mat = as.matrix(dist(ace.tip))
    Rvar[i] <- sum(mat[row(mat) == col(mat) + 1])
  }
  abs(1 - (var(Rvar) + (mean(as.matrix(y))/mean(y.hat))))
}
logLik(mle2(optL, start = list(lambda = 10), method = "L-BFGS-B"))->LL1
mle2(optL, start = list(lambda = 10), method = "L-BFGS-B")->f1
profile(f1,maxsteps=5000,prof.upper=10,trace=TRUE)->pf1
plot(pf1)
```

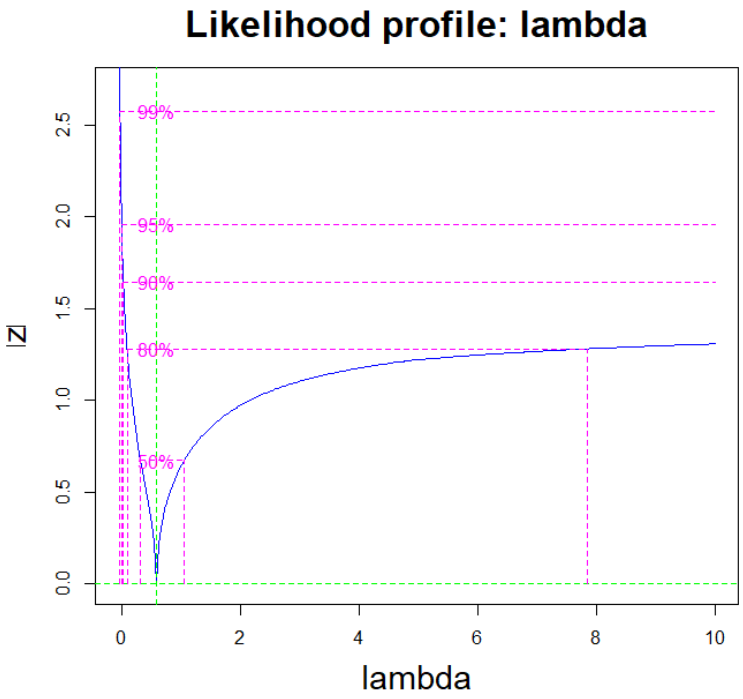

Figure 3. The likelihood profile of the *optL* function, used within *RRphylo* for the purpose of penalization.

As with version 2.2.0, *RRphylo* can perform multiple regression. If a predictor is specified by the argument 'x1' the the partial regression coefficient calculated for 'x1' is reported as $x1.rate.

The effect of $x1.rate is easy to assess graphically.

```
## given a tree (tree) and a phenotype (y) produce a predictor 'x1' as:
## create a predictor x1 related to y
# x1 <- (y+(runif(length(y), -0.5*sd(y), 0.5*sd(y))))*-1
# plot(y,x1) ## this plot shows the original relationship between y and
the predictor x1

## compute ancestral state estimates for x1
```

```
# RRphylo(tree, x1)$aces[,1]->ace1
## perform multiple regression RRphylo
# RRphylo(tree,y,x1=c(ace1,x1))->RR

# plot(y,RR$x1.rate*x1) ## this plot shows the effect of the partial
regression coefficient $x1.rate on the relationship between y and the
predictor x1
```

2. Locating shifts with the `search.shift` function

Title *Locating shifts in phenotypic evolutionary rates*

Description

The function *search.shift* tests whether individual clades or isolated tips dispersed through the phylogeny evolve at different *RRphylo* rates as compared to the rest of the tree. Instances of rate shifts may be automatically found.

Usage

```
search.shift(RR,status.type=c("clade","sparse"),node=NULL,
state=NULL,cov= NULL,nrep=1000,f=NULL,foldername)
```

Arguments

RR an object fitted by *RRphylo*.

status.type whether the “clade” or “sparse” condition must be tested**.**

node under the "clade" condition, the node (clade) to be tested for the rate shift. When multiple nodes are tested, they need to be written as in the example below. If 'node' is left unspecified, the function performs under the 'auto-recognize' feature, meaning it will automatically test individual clades for deviation of their rates from the background rate of the rest of the tree (see details).

state the state of the tips specified under the “sparse” condition.

cov the covariate to be indicated if its effect on rate values must be accounted for. Contrary to *RRphylo*, 'cov' needs to be as long as the number of tips of the tree.

nrep the number of simulations to be performed for the rate shift test, by default 'nrep' is set at 1000.

f the size of the smallest clade to be tested. By default, nodes subtending to one tenth of the tree tips are tested.

foldername the path of the folder where plots are to be found.

Details

The function *search.shift* takes the object produced by *RRphylo*. Two different conditions of rate change can be investigated. Under the "clade" condition the vector of node or nodes subjected to the shift must be provided. Alternatively, under the "sparse" case the (named) vector of states (indicating which tips are or are not evolving under the rate shift according to the tested hypothesis) must be indicated. In the "clade" case, the function may perform an 'auto-recognize' feature. Under such specification, the function automatically tests individual clades (from clades as large as one half of the tree down to a specified clade size) for

deviation of their rates from the background rate of the rest of the tree, which is identical to the "clade" case. An inclusive clade with significantly high rates is likely to include descending clades with similarly significantly high rates. Hence, with 'auto-recognize' the *search.shift* function is written as to scan clades individually and to select only the node subtending to the highest difference in mean absolute rates as compared to the rest of the tree. A plot of the tree highlighting nodes subtending to significantly different rates is automatically produced. Caution must be put into interpreting the 'auto-recognize' results. For instance, a clade with small rates and another with large rates could be individuated even under BM. This does not mean these clades are actual instances for rate shifts. Clades must be tested on their own without the 'auto-recognize' feature, which serves as guidance to the investigator, when no strong a priori hypothesis to be tested is advanced. The 'auto-recognize' feature is not meant to provide a test for a specific hypothesis. It serves as an optional guidance to understand whether and which clades show significantly large or small rates as compared to the rest of the tree. Individual clades are tested at once, meaning that significant instances of rate variation elsewhere on the tree are ignored. Conversely, running the "clade" condition without including the 'auto-recognize' feature, multiple clades presumed to evolve under the same shift are tested together, meaning that their rates are collectively contrasted to the rest of the tree, albeit they can additionally be compared to each other upon request. Under both the "clade" and "sparse" conditions, multiple clades could be specified at once, and optionally tested individually (for deviation of rates) against the rates of the rest of the tree and against each other. The histogram of random differences of mean rates distribution along with the position of the real difference of means is automatically generated by *search.shift*. Regardless of which condition is specified, the function output produces the real difference of means, and their significance value.

Value

Under all circumstances, *search.shift* provides a vector of **$rates**. If 'cov' values are provided, rates are regressed against the covariate and the residuals of such regression form the vector **$rates**. Otherwise, **$rates** are the same rates as with the RR argument.

Under "clade" case without specifying nodes (i.e. 'auto-recognize') a list including:

$all.clades for each detected node, the data-frame includes the average rate difference (computed as the mean rate over all branches subtended by the node minus the average rate for the rest of the tree) and the probability that it do represent a real shift. Probabilities are contrasted to simulations shuffling the rates across the tree branches for a number of replicates specified by the argument 'nrep'. Note that the p-values refer to the number of times the real average rates are larger (or smaller) than the rates averaged over the rest of the tree, divided by the number of simulations. Hence, large rates are significantly larger than the rest of the tree (at alpha = 0.05), when the probability is > 0.975; and small rates are significantly small for $p < 0.025$.

$single.clades the same as with 'all.clades' but restricted to the largest/smallest rate values along a single clade (i.e. nested clades with smaller rate shifts are excluded). Large rates are significantly larger than the rest of the tree (at alpha = 0.05), when the probability is > 0.975; and small rates are significantly small for $p < 0.025$.

Under "clade" condition by specifying the node argument:

$all.clades.together if more than one node is tested, this specifies the average rate difference and the significance of the rate shift, by considering all the specified nodes as evolving under a single rate. As with the 'auto-recognize' feature, large rates are significantly

larger than the rest of the tree (at alpha = 0.05), when the probability is > 0.975; and small rates are significantly small for p < 0.025.

$single.clades this gives the significance for individual clades, tested separately. As previously, large rates are significantly larger than the rest of the tree (at alpha = 0.05), when the probability is > 0.975; and small rates are significantly small for p < 0.025.

Under the "sparse" condition:

$state.results for each state, the data-frame includes the average rate difference (computed as the mean rate over all leaves evolving under a given state, minus the average rate for each other state or the rest of the tree) and the probability that the shift is real. Large rates are significantly larger (at alpha = 0.05), when the probability is > 0.975; and small rates are significantly small for p < 0.025. States are compared pairwise.

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
### DataOrnithodirans$massdino->massdino
### DataOrnithodirans$statedino->statedino
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
### DataOrnithodirans$massdino->massdino
### DataOrnithodirans$statedino->statedino
# RRphylo(tree=treedino,y=massdino)->dinoRates

## Case 1. Without accounting for the effect of a covariate
```

```
## Case 1.1 "clade" condition
## with auto-recognize
# search.shift(RR=dinoRates,status.type="clade",
#foldername=tempdir())

## testing two hypothetical clades
# search.shift(RR=dinoRates,status.type="clade",node=c(697,748),
# foldername=tempdir())

## Case 1.2 "sparse" condition

## testing the sparse condition.#
search.shift(RR=dinoRates,status.type="sparse",
# state=statedino,foldername=tempdir())

## Case 2. Accounting for the effect of a covariate

## Case 2.1 "clade" condition
# search.shift(RR=dinoRates,status.type="clade",
# cov=massdino,foldername=tempdir())

## Case 2.2 "sparse" condition
# search.shift(RR=dinoRates,status.type="sparse",state=statedino,
```

```
# cov=massdino,foldername=tempdir())
```

3. Producing phenotypes with desired trends in rate over time or phenotypic mean over time with `setBM`

Title *Producing simulated phenotypes with trends*

Description

The function setBM is wrapper around 'phytools' *fastBM* function, which generates BM simulated phenotypes with or without a trend.

Usage

```
setBM(tree,nY=1,s2=1,a=0,type=c("","brown","trend","drift"),
trend.type=c("linear","stepwise"),tr=10,t.shift=0.5,es=2,ds=1)
```

Arguments

tree a phylogenetic tree.

nY the number of phenotypes to simulate.

s2 value of the Browian rate to use in the simulations.

a the phenotype at the tree root.

type the type of phenotype to simulate. With the option "brown" the phenotype will have no trend in the phenotypic mean or in the rate of evolution. A variation in the phenotypic mean

over time (a phenotypic trend) is obtained by selecting the option “drift”. A trend in the rate of evolution produces an increased variance in the residuals over time. This is obtained by specifying the option “trend”.

trend.type two kinds of heteroscedastic residuals are generated under the “trend” type. The option “linear” produces an exponential increase (or decrease) in heteroscedasticity, whereas the “stepwise” option produces an increase (or decrease) after a specified point in time.

tr the intensity of the trend with the “stepwise” option is controlled by the 'tr' argument. The scalar 'tr' is the multiplier of the branches extending after the shift point as indicated by ' t.shift'.

t.shift the relative time distance from the tree root where the stepwise change in the rate of evolution is indicated to apply.

es when 'trend.type'="linear", 'es' is a scalar representing the exponent at which the evolutionary time (i.e. distance from the root) scales to change to phenotypic variance. With 'es' = 1 the phenotypic rate will be trendless, with 'es' < 1 the variance of the phenotypes will decrease exponentially towards the present and the other way around with 'es' > 1.

ds a scalar indicating the change in phenotypic mean in the unit time, in 'type'="drift"

case. With 'ds' = 0 the phenotype will be trendless, with 'ds' < 0 the phenotypic mean will decrease towards the present and the other way around with 'ds' > 0.

Details

setBM differs from *fastBM* in that the produced phenotypes are checked for the existence of a temporal trend in the phenotype. The user may specify whether she wants trendless data (option “brown”), phenotypes trending in time (option “drift”), or phenotypes whose variance increases/decreases exponentially over time, consistently with the existence of a trend in the

rate of evolution (option “trend”). In the latter case, the user may indicate the intensity of the trend (by applying different values of 'es'), and whether it should occur after a given proportion of the tree height (hence a given point back in time, specified by the argument 't.shift').

Trees in *setBM* are treated as non ultrametric. If an ultrametric tree is fed to the function, *setBM* alters slightly the leaf lengths multiplying randomly half of the leaves by $1 * 10^{-3}$ in order to make it non-ultrametric.

Value

Either an object of class array containing a single phenotype or an object of class matrix of *n* phenotypes as columns, where *n* is indicated as 'nY' = n

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino

# setBM(tree=treedino, nY= 1, type="brown")
# setBM(tree=treedino, nY= 1, type="drift", ds=2)
# setBM(tree=treedino, nY= 1, type="trend", trend.type="linear", es=2)
```

4. `sizedsubtree`

Title *Find a node subtending to a clade of desired size*

Description

The function *sizedsubtree* scans a phylogentic tree to randomly find nodes subtending to a subtree of desired minimum size, up to one half of the tree size (number of tips).

Usage

```
sizedsubtree(tree,Size=NULL, time.limit=10)
```

Arguments

tree a phylogenetic tree

Size the desired size of the tree subtending to the extracted node. By default, the minimum tree size is set at one tenth of the tree size (i.e. number of tips).

time.limit specifies a limit to the searching time, a warning message is thrown if the limit is reached.

Details

The argument 'time.limit' sets the searching time. The algorithm stops if that limit is reached, avoiding recursive search when no solution is in fact possible.

Value

A node subtending to a subtree of desired minimum size.

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
```

```
# sizedsubtree(tree=treedino)


# library(geiger)
# tips(treedino,sizedsubtree(treedino,size=50))
## to view the tip extracted
# extract.clade(treedino,sizedsubtree(treedino,size=50))
## to extract the subtree
```

5. `evo.dir`

Title *Phylogenetic vector analysis of phenotypic change*

Description

This function quantifies direction, size and rate of evolutionary change of multivariate traits along node-to-tip paths and between species.

Usage

```
evo.dir(RR,angle.dimension=c("rates","phenotypes"),y.type=c("original",
"RR"),y=NULL,pair.type=c("node","tips"),pair=NULL,node=NULL,
random=c("yes","no"),nrep=100)
```

Arguments

RR an object produced by *RRphylo*.

angle.dimension specifies whether “rates” or “phenotypes” vectors are used.

y.type must be indicated when 'angle.dimension' = “phenotypes”. If “original”, it uses the phenotypes provided by the user, if “RR” it uses RR$predicted.phenotypes as variable.

y specifies the phenotypes to be provided if 'y.type' = “original”.

pair.type either “node” or “tips”. Angles are computed between each possible couple of species descending from a specified node (“node”), or between a given couple of species (“tips”).

pair species pair to be specified if 'pair.type' = “tips”. It needs to be written as in the example below.

node node number to be specified if 'pair.type' = “node”. Notice the node number must refer to the dichotomic version of the original tree, as produced by *RRphylo*.

random whether to perform randomization test (“yes”/“no”).

nrep number of replications must be indicated if 'random' = “yes”. It is set at 100 by default.

Details

The way *evo.dir* computes vectors depends on whether phenotypes or rates are used as variables. *RRphylo* rates along a path are aligned along a chain of ancestor/descendant relationships. As such, each rate vector origin coincides to the tip of its ancestor, and the resultant of the path is given by vector addition. In contrast, phenotypic vectors are computed with reference to a common origin (i.e. the consensus shape in a geometric morphometrics). In this latter case, vector subtraction (rather than addition) will define the resultant of the species evolutionary direction.

It is important to realize that resultants could be at any angle even if the species (the terminal vectors) have similar phenotypes, because path resultants, rather than individual phenotypes,

are being contrasted. However, the function also provides the angle between individual phenotypes as 'angle.between.species'.

To perform randomization test, the evolutionary directions of the two species are collapsed together. Then, for each variable, the median is found, and random paths of the same size as the original paths are produced sampling at random from the 47.5^{th} to the 52.5^{th} percentile around the medians. This way, a random distribution of angles is obtained under the hypothesis that the two directions are actually parallel. The 'angle.direction' represents the angle formed by the species phenotype and a vector of 1s (as long as the number of variables representing the phenotype). This way, each species phenotype is contrasted to the same vector. The 'angle.direction' values could be inspected to test whether individual species phenotypes evolve towards similar directions.

Value

Under all specs, *evo.dir* returns a list object. The length of the list is one if 'pair.type' = "tips". If 'pair.type' = "node", the list is as long as the number of all possible species pairs descending from the node. Each element of the list contains:

angle.path.A angle of the resultant vector of species A to MRCA

vector.size.species.A size of the resultant vector of species A to MRCA

angle.path.B angle of the resultant vector of species B to MRCA

vector.size.species.B size of the resultant vector of species B to MRCA

angle.between.species.to.mrca angle between the species paths resultant vectors to the MRCA

angle.between.species angle between species vectors (as they are, without computing the path)

MRCA the node identifying the most recent common ancestor of A and B

angle.direction.A angle of the vector of species A (as it is, without computing the path) to a fixed reference vector (the same for all species)

vec.size.direction.A size of the vector of species A

angle.direction.B angle of the vector of species B (as it is, without computing the path) to a fixed reference vector (the same for all species)

vec.size.direction.B size of the vector of species B

If 'random' = "yes", results also include p-values for the angles.

Examples

```
### data("DataApes")

### DataApes$PCstage->PCstage

### DataApes$Tstage->Tstage


# RRphylo(tree=Tstage,y=PCstage)->RR


## Case 1. Without performing randomization test

## Case 1.1 Computing angles between rate vectors

## for each possible couple of species descending from node 72

# evo.dir(RR,angle.dimension="rates",pair.type="node",node=72,

# random="no")


## for a given couple of species

# evo.dir(RR,angle.dimension="rates",pair.type="tips",
```

```
# pair=c("Sap_1","Tro_2"),random="no")

## Case 1.2 computing angles between phenotypic vectors provided by
## the user for each possible couple of species descending from node
# 72
# evo.dir(RR,angle.dimension="phenotypes",y.type="original",
# y=PCstage,pair.type="node",node=72,random="no")

## for a given couple of species
# evo.dir(RR,angle.dimension="phenotypes",y.type="original",
y=PCstage,pair.type="tips",pair=c("Sap_1","Tro_2"),random="no")

## Case 1.3 computing angles between phenotypic vectors produced by
## RRphylo for each possible couple of species descending from node
## 72
# evo.dir(RR,angle.dimension="phenotypes",y.type="RR",
pair.type="node",node=72,random="no")

## for a given couple of species
# evo.dir(RR,angle.dimension="phenotypes",y.type="RR",
# pair.type="tips",pair=c("Sap_1","Tro_2"),random="no")


## Case 2. Performing randomization test
```

```
## Case 2.1 Computing angles between rate vectors
## for each possible couple of species descending from node 72
# evo.dir(RR,angle.dimension="rates",pair.type="node",node=72,
# random="yes",nrep=100)

## for a given couple of species
# evo.dir(RR,angle.dimension="rates",pair.type="tips",
# pair=c("Sap_1","Tro_2"),random="yes",nrep=100)

## Case 2.2 computing angles between phenotypic vectors provided by
## the user for each possible couple of species descending from node
## 72

# evo.dir(RR,angle.dimension="phenotypes",y.type="original",
y=PCstage,pair.type="node",node=72,random="yes",nrep=100)


## for a given couple of species
# evo.dir(RR,angle.dimension="phenotypes",y.type="original",
# y=PCstage,pair.type="tips",pair=c("Sap_1","Tro_2"),
# random="yes",nrep=100)
```

```
## Case 2.3 computing angles between phenotypic vectors produced by
## RRphylo for each possible couple of species descending from node
## 72
# evo.dir(RR,angle.dimension="phenotypes",y.type="RR",
# pair.type="node",node=72,random="yes",nrep=100)

## for a given couple of species
# evo.dir(RR,angle.dimension="phenotypes",y.type="RR",
# pair.type="tips",pair=c("Sap_1","Tro_2"),random="yes",nrep=100)
```

6. `retrieve.angles`

Title *Extracting a user-specified subset of the evo.dir results*

Description

This function takes the result list produced by *evo.dir* as the input and extracts a specific subset of it. The user may specify whether to extract the set of angles between species resultant vectors and the MRCA, the size of resultant vectors, or the set of angles between species.

Usage

```
retrieve.angles(angles.res,wishlist=c("anglesMRCA","angleDir",
"angles.between.species"),random=c("yes","no"),focus=c("node","species",
"both","none"),node=NULL,species=NULL,write2csv=c("yes","no"))
```

Arguments

angles.res the object resulting from *evo.dir* function.

wishlist specifies whether to extract angles and sizes ("anglesMRCA") of resultant vectors between individual species and the MRCA, angles and sizes ("angleDir") of vectors between individual species and a fixed reference vector (the same for all species), or angles between species resultant vectors ("angles.between.species").

random it needs to be "yes" if angles.res object contains randomization results.

focus it can be "node", "species", "both", or "none", whether the user wants the results for a focal node, or for a given species, for both, or just wants to visualize everything.

node must be indicated if 'focus' = "node" or "both". As for *evo.dir*, the node number must refer to the dichotomic version of the original tree, as produced by *RRphylo*.

species must be indicated if 'focus' = "species" or "both".

write2csv if "yes" results are saved to a .csv file in your working directory.

Details

retrieve.angles allows to focalize the extraction to a particular node, species, or both. Otherwise it returns the whole dataset.

Value

retrieve.angles outputs an object of class 'data.frame'.

If 'wishlist' = "anglesMRCA", the data frame includes:

MRCA the most recent common ancestor the angle is computed to.

species species ID.

angle the angle between the resultant vector of species and the MRCA.

vector.size the size of the resultant vector computed from species to MRCA.

If 'wishlist' = “angleDir”, the data frame includes:

MRCA the most recent common ancestor the vector is computed to.

species species ID.

angle.direction the angle between the vector of species and a fixed reference.

vector.size the size of the vector of species.

If 'wishlist' = “angles.between species”, the data frame includes:

MRCA the most recent common ancestor the vector is computed from.

species pair IDs of the species pair the “angle between species” is computed for.

angleBTWspecies2MRCA angle between species resultant vectors to MRCA

anglesBTWspecies angle between species resultant vectors

Examples

```
### data("DataApes")

### DataApes$PCstage->PCstage

### DataApes$Tstage->Tstage


# RRphylo(tree=Tstage,y=PCstage)->RR


## Case 1. “evo.dir” without performing randomization

# evo.dir(RR,angle.dimension="rates",pair.type="node",node=57,

# random="no")->evo.p
```

```
## Case 1.1 retrieve angles and sizes of resultant vectors between
individual species and the MRCA:
## for a focal node
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="no",
# focus="node", node=68,write2csv="no")

## for a fcal species
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="no",focus=
# "species", species="Sap", write2csv="no")

## for both focal node and species
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="no",
# focus="both", node=68,species="Sap",write2csv="no")

## without any specific requirement
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="no",
# focus="none", write2csv="no")

## Case 1.2 retrieve angles and sizes of vectors between individual
species and a fixed reference vector:
## for a focal node
# retrieve.angles(evo.p,wishlist="angleDir",random="no",
# focus="node", node=68,write2csv="no")
```

```
## for a focal species
# retrieve.angles(evo.p,wishlist="angleDir",random="no",
# focus="species", species="Sap", write2csv="no")

## for both focal node and species
# retrieve.angles(evo.p,wishlist="angleDir",random="no",
# focus="both", node=68,species="Sap",write2csv="no")

## without any specific requirement
# retrieve.angles(evo.p,wishlist="angleDir",random="no",
# focus="none", write2csv="no")

## Case 1.3 retrieve angles between species resultant vectors:
## for a focal node
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="no", focus="node", node=68,write2csv="no")

## for a focal species
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="no", focus="species", species="Sap", write2csv="no")
## for both focal node and species
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="no", focus="both",node=68,species="Sap",write2csv="no")
```

```
## without any specific requirement
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="no", focus="none",write2csv="no")


## Case 2. "evo.dir" with performing randomization
# evo.dir (RR,angle.dimension="rates",pair.type="node",node=57,
# random="yes")->evo.p

## Case 2.1 retrieve angles and sizes of resultant vectors between
## individual species and the MRCA: for a focal node
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="yes",
# focus="node",node=68,write2csv="no")

## for a focal species
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="yes", focus=
# "species", species="Sap", write2csv="no")

## for both focal node and species
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="yes",focus=
# "both",node=68,species="Sap",write2csv="no")

## without any specific requirement
```

```
# retrieve.angles(evo.p,wishlist="anglesMRCA",random="yes",focus=
# "none", write2csv="no")

## Case 2.2 retrieve angles and sizes of resultant vectors between
individual species and the MRCA:
## for a focal node
# retrieve.angles(evo.p,wishlist="angleDir",random="yes",
# focus="node",node=68,write2csv="no")

## for a focal species
# retrieve.angles(evo.p,wishlist="angleDir",random="yes",focus=
# "species", species="Sap", write2csv="no")

## for both focal node and species
# retrieve.angles(evo.p,wishlist="angleDir",random="yes",focus=
# "both", node=68, species="Sap",write2csv="no")

## without any specific requirement
# retrieve.angles(evo.p,wishlist="angleDir",random="yes",focus=
# "none", write2csv="no")

## Case 2.3 retrieve angles between species resultant vectors:

## for a focal node
```

```
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="yes",focus="node", node=68,write2csv="no")

## for a focal species
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="yes", focus="species", species="Sap", write2csv="no")

## for both focal node and species
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="yes", focus="both",node=68,species="Sap",write2csv="no")

## without any specific requirement
# retrieve.angles(evo.p,wishlist="angles.between.species",
# random="yes", focus="none",write2csv="no")
```

7. `angle.matrix`

Title *Ontogenetic shape vectors analysis*

Description

This function computes ontogenetic vectors for each species and compares them between species pairs.

Usage

```
angle.matrix(RR,node,Y=NULL,select.axes=c("no","yes"),
type=c("phenotypes","rates"),cova=NULL)
```

Arguments

RR an object produced by *RRphylo*.

node the number of the node subtending species to compute ontogenetic vectors on. Notice the node number must refer to the dichotomic version of the original tree, as it is produced by *RRphylo*.

Y multivariate trait values at tips.

select.axes if “yes”, 'Y' variables are individually regressed against developmental stages and only significant phenotypes are retained to compute ontogenetic vectors. If “no”, all variables are retained.

type specifies weather to perform the analysis on phenotypic (“phenotypes”) or rate (“rates”) vectors.

cova the covariate to be indicated if its effect on rate values must be accounted for. Contrary to *RRphylo*, 'cova' needs to be as long as the number of tips of the tree. As the covariate only affects rates computation, there is no covariate to provide when 'type' = "phenotypes".

Details

The *angle.matrix* function takes as objects a phylogenetic tree (deriving directly from an RR object), including the different ontogenetic stages of each species as polytomies. Names at tips must be written as species ID and stage number separated by the underscore (for example “Sap_1”, that is *Homo sapiens* at stage 1).

The RRphylo object *angle.matrix* is fed with is just used to extract the dichotomized version of the phylogeny. This is useful because node number changes randomly at dichotomizing non-binary trees. However, when performing *angle.matrix* with the covariate there is no need for the RR object to be produced with the argument covariate set to TRUE. Furthermore, as the covariate only affects the rates computation, it makes no sense to use it when computing vectors for phenotypic variables.

Once angles and vectors are computed, *angle.matrix* performs two tests by means of standard major axis (SMA) regression. For each species pair, the "biogenetic test" verifies whether the angle between species grows during development, meaning that the two species becomes less similar to each other during growth, in keeping with the biogenetic law. The "paedomorphosis test" tells whether there is heterochronic shape change in the data. Under paedomorphosis, the adult stages of one (paedomorphic) species will resemble the juvenile stages of the other (peramorphic) species. The test regresses the angles formed by the shapes at different ontogenetic stages of a species to the shape at the youngest stage of the other in the pair, against age. Then, it tests whether the two regression lines (one per species) have different slopes, and whether they have different signs. If the regression lines point to different directions and differ from each other, it means that one of the two species in the pair resembles, with age, the juveniles of the other, indicating paedomorphosis. Ontogenetic vectors of individual species are further computed, in reference to the MRCA of the pair, and to the first stage of each species (i.e. intraspecifically). Importantly, the size of the ontogenetic vectors of rates tell whether the two species differ in terms of developmental rate, which is crucial to understand which process is behind paedomorphosis, where it applies.

While performing the analysis, the function prints messages on-screen. If 'select.axes' = "yes", it informs the user about which phenotypic variables are used. Secondly, it specifies

whether ontogenetic vectors to MRCA, and intraspecific ontogenetic vectors significantly differ in angle or size between species pairs. Then, for each species pair, it indicates if the biogenetic law and paedomorphosis apply.

Value

A list containing 4 objects**.**

$regression.matrix: a list object including "angles between species" and "angles between species to MRCA" matrices for all possible combinations of species pairs from the two sides descending from the MRCA. For each matrix, corresponding biogenetic and paedomorphosis tests are reported.

$`angles between species`

$`Species1/Species2`

$matrix

$paedomorphosis test

$biogenetic test

$`Species1/Species3`

$matrix

$paedomorphosis test

$biogenetic test

$`angles between species 2 MRCA`

$`Species1/Species2`

$matrix

$paedomorphosis test

$biogenetic test

$`Species1/Species3`

$matrix

$paedomorphosis test

$biogenetic test

$angles.2.MRCA.and.vector.size: a data.frame including angles between the resultant vector of species and the MRCA and the size of the resultant vector computed from species to MRCA, per stage per species.

$ontogenetic.vectors2MRCA: a data.frame including angle, size, and corresponding x and y components, of ontogenetic vectors between each species and the MRCA. For both angle and size, the p-value for the difference between species pairs is reported.

$ontogenetic.vectors.to.1st.stage: a list object containing:

$matrices: for all possible combinations of species pairs from the two sides descending form the MRCA, the upper triangle of the matrix contains angles between different ontogenetic stages for the first species (i.e. Species1). The same applies to the lower triangle for the second species (i.e. Species2 or Species3).

$ `Species1/Species2`

$ `Species1/Species3`

$vectors: for all possible combinations of species pairs from the two sides descending form the MRCA, angle and size of ontogenetic vectors computed to the first stage of each species. For both angle and size, the p-value for the difference between the species pair is reported.

$ `Species1/Species2`

$ `Species1/Species3`

Examples

```
### data("DataApes")

### DataApes$PCstage->PCstage

### DataApes$Tstage->Tstage

### DataApes$CentroidSize->CS


### RRphylo(tree=Tstage,y=PCstage)->RR


## Case 1. "angle.matrix" without accounting for te effect of a

## covariate


## Case 1.1 selecting only shape variables that show significant

## relationship with age on phenotypic vectors

# angle.matrix(RR,node=72,Y=PCstage,select.axes="yes",

# type="phenotypes")


## on rates vectors

# angle.matrix(RR,node=72,Y=PCstage,select.axes="yes",

# type="rates")


## Case 1.2 using all shape variables
```

```
## on phenotypic vectors
# angle.matrix(RR,node=72,Y=PCstage,select.axes="no",
# type="phenotypes")

## on rates vectors
# angle.matrix(RR,node=72,Y=PCstage,select.axes="no",type="rates")

## Case 2. "angle.matrix" accounting for the effect of a covariate
## (on rates vectors only)

## Case 2.1 selecting only shape variables that show significant
## relationship with age
# angle.matrix(RR,node=72,Y=PCstage,select.axes="yes",
# type="rates", cova=CS)

## Case 2.2 using all shape variables
# angle.matrix(RR,node=72,Y=PCstage,select.axes="no",
# type="rates", cova=CS)
```

8. makeL

Title *Matrix of branch lengths along root-to-tip paths*

Description

This function produces a *n x m* matrix, where n=number of tips and m=number of branches (i.e. n + number of nodes). Each row represents the branch lengths along a root-to-tip path.

Usage

```
makeL(tree)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

Value

The function returns a *n x m* matrix of branch lengths along a root-to-tip path.

Examples

```
### data("DataApes")

### DataApes$Tstage->Tstage

# makeL(tree=Tstage)
```

9. `makeL1`

Title *Matrix of branch lengths along a root-to-node path*

Description

This function produces a *n x n* matrix, where n=number of internal branches. Each row represents the branch lengths along a root-to-node path.

Usage

```
makeL1(tree)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

Value

The function returns a *n x n* matrix of branch lengths along a root-to-node path.

Examples

```
### data("DataApes")

### DataApes$Tstage->Tstage

# makeL1(tree=Tstage)
```

10. `getMommy`

Title *Upward tip or node to root path*

Description

This function is a wrapper around phytools *getDescendants* (Revell 2012). It returns the node path from a given node or species to the root of the phylogeny.

Usage

```
getMommy(tree,N,curr=NULL)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

N the number of node or tip to perform the function on. Notice the function only works with number, not tip labels.

curr has not to be provided by the user.

Details

The object 'curr' is created inside the function in order to produce an array of nodes on the path.

Value

The function produces a vector of node numbers as integers, collated from a node or a tip towards the tree root.

Examples

```
### data("DataApes")
### DataApes$Tstage->Tstage
# getMommy(tree=Tstage,12)
```

11. makeFossil

Title *make fossil species on a phylogeny*

Description

This function takes an object of class 'phylo' and randomly changes the lengths of the leaves.

Usage

```
makeFossil(tree,p=0.5,ex=0.5)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

p the proportion of tips involved. By default it is half of the number of tips.

ex the multiplying parameter to change the leaf lengths. It is set at 0.5 by default.

Value

The function produces a phylogeny having the same backbone of the original one.

Examples

```
### data("DataApes")

### DataApes$Tstage->Tstage


# makeFossil(tree=Tstage,p=0.5,ex=0.5)
```

12. `swap.phylo`

Title *test the effect of phylogenetic uncertainty on rate shifts found at a particular node*

Description

The function uses a number of alternative phylogenies with altered (as compared to the reference tree) topology and branch lengths to tests whether the tips descending from the specified node ('node') have statistically different rates from the rest of the tree. A phenotypic vector 'y' must be supplied. Eventually, the effect of a covariate could be included.

Usage

```
swap.phylo(tree,si=0.5,si2=0.5,node,y,rts,nrep=100,cov=NULL)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

si the proportion of tips whose topologic arrangement will be swapped.

si2 the proportion of nodes whose age will be changed.

node the focal node to be tested.

y the phenotype under testing.

rts the rates found by *RRphylo* on the original tree.

nrep the number of simulated trees to be produced.

cov the covariate to be indicated if its effect on rate values must be accounted for. Contrary to *RRphylo*, 'cov' needs to be as long as the number of tips of the tree.

Details

swap.phylo changes the tree topology and branch lengths to a level specified by the user. Up to half of the tips, and half of the branch lengths can be changed randomly. The function provides a ”swapped” tree, yet, importantly, once a shift in the rate of evolution has been found by *RRphylo*, this function can be used to test whether the shift depends on the tree topology and branch lengths. It runs *RRphylo* on swapped trees (default is 100) and then calculates the absolute rate difference between all the branches of the shifted node and the rest of the tree. A t-test is eventually performed to assess significance.

Value

The function returns a list object containing:

$p.swap the probability that the rates at 'node' are different from rates at the rest of the tree.

$rates the distribution of rates per branch as calculated by *RRphylo* on ”swapped” phylogenies.

Examples

```
### data("DataApes")

### DataApes$PCstage->PCstage

### DataApes$Tstage->Tstage

### DataApes$CentroidSize->CS


## Case 1. swap.phylo without accounting for the effect of a

## covariate

# RRphylo(tree=Tstage,y=PCstage)->RR

# RR$rates->rr
```

```
# swap.phylo(tree=Tstage,node=72,y=PCstage,rts=rr)


## Case 2. swap.phylo accounting for the effect of a covariate
# RRphylo(tree=Tstage,y=CS)->RRcova
# c(RRcova$aces,CS)->cov.values
# c(rownames(RRcova$aces),names(CS))->names(cov.values)
# RRphylo(tree=Tstage,y=PCstage,cov=cov.values)->RR
# RR$rates->rr


# swap.phylo(tree=Tstage,node=72,y=PCstage,rts=rr,cov=CS)
```

13. swapONE

Title *create alternative phylogenies from a given tree*

Description

The function produces an alternative phylogeny with altered topology and branch lengths, and computes the Kuhner-Felsenstein (Kuhner & Felsenstein 1994) distance between original and "swapped" tree.

Usage

```
swapONE(tree,node=NULL,si=0.5,si2=0.5,plot.swap=FALSE)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

node if specified, the clades subtended by such node(s) are imposed to be monophyletic. In this case, the function can still swap tips *within* the clade.

si the proportion of tips whose topologic arrangement will be swapped.

si2 the proportion of nodes whose age will be changed.

plot.swap if TRUE, the function plots the swapped tree. Swapped positions appear in red. Nodes with altered ages appear in green.

Details

swapONE changes the tree topology and branch lengths. Up to half of the tips, and half of the branch lengths can be changed randomly. Each randomly selected node is allowed to move up to 2 nodes apart from its original position.

Value

The function returns a list containing the "swapped" version of the original tree, and the Kuhner-Felsenstein distance between the trees.

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino

## Case 1. Change the topology and the branch lengths for the entire
## tree
```

```
# swapONE(tree=treedino,si=0.5,si2=0.5,plot.swap=FALSE)

## Case 2. Change the topology and the branch lengths of the
## tree by keeping the monophyly of a specific clade

# swapONE(tree=treedino,node=423,si=0.5,si2=0.5,plot.swap=FALSE)
```

14. plotRates

Title *plot RRphylo rates at a specified node*

Usage

```
plotRates(RR,node,export.tiff=TRUE,foldername)
```

Description

The function *plotRates* plots the *RRphylo* rates computed for a given clade as compared to the rates computed for the rest of the tree.

Arguments

RR an object produced by *RRphylo*.

node the node subtending the clade of interest.

export.tiff if TRUE the function save a "rate_bars.tiff" file inside the working directory. It is TRUE by default.

foldername the path of the folder where plots are to be found.

Value

The function produces two barplots. On the left side, the rates (in absolute values) computed for the focal clade (blue) are plotted against the rates of the rest of the tree (green). On the right side, the absolute rates of individual branches of the focal clade are collated in increasing rate value (blue bars), and contrasted to the average rate computed over the rest of the tree branches (the vertical red line). It also returns the rate values for both nodes and species descending from the focal node.

Examples

```
### data("DataApes")

### DataApes$PCstage->PCstage

### DataApes$Tstage->Tstage# RRphylo(tree=Tstage,y=PCstage)->RR


# plotRates(RR,node=72,export.tiff = FALSE,foldernam=tempdir())
```

15. `search.trend`

Title *searching for evolutionary trends in phenotypes and rates*

Description

This function searches for evolutionary trends in phenotypic mean and evolutionary rates.

Usage

```
search.trend(RR,y,x1=NULL,nsim=100,clus=0.5,node=NULL,foldername,
cov=NUL,ConfInt=FALSE)
```

Arguments

RR an object produced by RRphylo.

y the named vector (or matrix if multivariate) of phenotypes.

x1 the additional predictor to be specified if the RR object has been created using an additional predictor (i.e. multiple version of *RRphylo*). 'x1' vector must be as long as the number of nodes plus the number of tips of the tree, which can be obtained by running *RRphylo* on the predictor as well, and taking the vector of ancestral states and tip values to form the 'x1'.

nsim number of simulations to be performed. It is set at 100 by default.

clus the proportion of clusters to be used in parallel computing.**node** number of individual clades to be specifically tested and contrasted to each other. It is NULL by default. Notice the node number must refer to the dichotomic version of the original tree, as produced by RRphylo.

cov the covariate values to be specified if the RR object has been created using a covariate for rates calculation. As for RRphylo, 'cov' must be as long as the number of nodes plus the number of tips of the tree, which can be obtained by running RRphylo on the covariate as well, and taking the vector of ancestral states and tip values to form the covariate (see the example below).

foldername the path of the folder where plots are to be found.

ConfInt if TRUE, the function returns 95% confidence intervals around phenotypes and rates produced according to the Brownian motion model of evolution. It is FALSE by default.

Details

The function simultaneously returns the regression of phenotypes and phenotypic evolutionary rates against age (in terms of distance from the tree root) tested against Brownian motion simulations to assess significance. It stores the rates (absolute values) versus age regression and the phenotype versus age regression plots as .pdf files. In the plots, the 95% confidence intervals of phenotypes and rates simulated under the Brownian motion for each node are plotted as shaded areas. Regression lines are printed for all regressions. To assess significance, slopes are compared to a family of simulated slopes (BM_{slopes}, where the number of simulations is equal to 'nsim'), generated under the Brownian motion, using the *fastBM* function in the package phytools.

Individual nodes are compared to the rest of the tree in different ways depending on whether phenotypes or rates versus age regressions are tested. With the former, the regression slopes for individual clades and the slope difference between clades is contrasted to slopes obtained through Brownian motion simulations. For the latter, regression models are tested and contrasted to each other referring to estimated marginal means, by using the *emmeans* function in the package emmeans.

Value

The function returns a list object including:

$rbt: for each branch of the tree, the *RRphylo* rates and the distance from the tree root (age). If 'y' is multivariate, it also includes the multiple rates for each 'y' vector. If 'node' is specified, each branch is classified as belonging or not to the indicated clades.

$pbt: a data frame of phenotypic values and their distance from the tree root for each node (i.e. ancestral states) and tip of the tree.

$phenotypic.regression: results of phenotype versus age regression. It reports a p-value for the regression slope between the variables (p.real), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), and a parameter indicating the deviation of the phenotypic mean from the root value in terms of the number of standard deviations of the trait distribution (dev). dev is 0 under Brownian Motion. Only p.random should be inspected to assess significance.

$rate.regression: results of the rates (absolute values) versus age regression. It reports a p-value for the regression between the variables (p.real), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), and a parameter indicating the ratio between the range of phenotypic values and the range of such values halfway along the tree height, divided to the same figure under Brownian motion (spread). spread is 1 under Brownian motion. Only p.random should be inspected to assess significance.

$ConfInts: the 95% confidence intervals around phenotypes and rates produced according to the Brownian motion model of evolution.

If specified, individual nodes are tested as the whole tree, the results are summarized in the objects:

$node.phenotypic.regression results of phenotype versus age regression through node. It reports the slope for the regression between the variables at node (slope), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), the difference between estimated marginal means predictions for the group and for the rest of the tree (emm.difference), and a pvalue for the emm.difference (p.emm).

$node.rate.regression results of the rates (absolute values) versus age regression through node. It reports the difference between estimated marginal means predictions for the group and for the rest of the tree (emm.difference), a p-value for the emm.difference (p.emm), the difference between regression slopes for the group and for the rest of the tree (slope.difference), and a p-value for the slope.difference (p.slope).

If more than one node is specified, the object **$group.comparison** reports the same results as $node.phenpotypic.regression and $node.rate.regression obtained by comparing individual clades to each other.

Examples

```
### data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
### DataOrnithodirans$massdino->massdino

## Extract Pterosaurs tree and data
# library(ape)
# extract.clade(treedino,748)->treeptero
# massdino[match(treeptero$tip.label,names(massdino))]->masspteroo
# masspteroo[match(treeptero$tip.label,
# names(masspteroo))]->masspteroo

## Case 1. "RRphylo" whitout accounting for the effect of a ##
covariate
# RRphylo(tree=treeptero,y=log(masspteroo))->RRptero
```

```
## Case 1.1. "search.trend" whitout indicating nodes to be tested
## for trends
# search.trend(RR=RRptero, y=log(massptero), nsim=100, clus=0.5,
# foldername=tempdir(),cov=NULL,ConfInt=FALSE,node=NULL)

## Case 1.2. "search.trend" indicating nodes to be specifically
## tested for trends
# search.trend(RR=RRptero, y=log(massptero), nsim=100, node=143,
# clus=0.5,foldername=tempdir(),cov=NULL,ConfInt=FALSE)

## Case 2. "RRphylo" accounting for the effect of a covariate
## "RRphylo" on the covariate in order to retrieve ancestral state
## values
# RRphylo(tree=treeptero,y=log(massptero))->RRptero
# c(RRptero$aces,log(massptero))->cov.values
# names(cov.values)<-c(rownames(RRptero$aces),names(massptero))
# RRphylo(tree=treeptero,y=log(massptero),
# cov=cov.values)->RRpteroCov

## Case 2.1. "search.trend" whitout indicating nodes to be tested
## for trends
# search.trend(RR=RRpteroCov, y=log(massptero), nsim=100,
# clus=0.5,foldername=tempdir(),ConfInt=FALSE,cov=cov.values)
```

```
## Case 2.2. "search.trend" indicating nodes to be specifically
## tested for trends
# search.trend(RR=RRpteroCov, y=log(massptero), nsim=100,
# node=143, clus=0.5,foldername=tempdir(),ConfInt=FALSE,
# cov=cov.values)

## Case 3. "search.trend" on multiple "RRphylo"
### data("DataCetaceans")
### DataCetaceans$treecet->treecet
### DataCetaceans$masscet->masscet
### DataCetaceans$brainmasscet->brainmasscet
### DataCetaceans$aceMyst->aceMyst
# drop.tip(treecet,treecet$tip.label[-match(names(brainmasscet),
# treecet$tip.label)])->treecet.multi
# masscet[match(treecet.multi$tip.label,
# names(masscet))]->masscet.multi
# RRphylo(tree=treecet.multi,y=masscet.multi)->RRmass.multi
# RRmass.multi$aces[,1]->acemass.multi
# c(acemass.multi,masscet.multi)->x1.mass
# RRphylo(tree=treecet.multi,y=brainmasscet,x1=x1.mass)->RRmulti

# search.trend(RR=RRmulti, y=brainmasscet,x1=x1.mass,clus=0.5,
# foldername=tempdir())
```

16. `getSis`

Title *Get sister clade*

Description

The function identifies and returns the sister clade of a given node/tip.

Usage

```
getSis(tree,n,printZoom= TRUE)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

n number of focal node or name of focal tip.

printZoom if TRUE the function plots the tree section of interest.

Value

The sister node number or sister tip name. In case of polytomies, the function returns a vector.

Examples

```
### data(DataOrnithodirans)

### DataOrnithodirans$treedino->treedino
```

```
# getSis(tree=treedino,n=678,printZoom=FALSE)
# getSis(tree=treedino,n="Shenzhoupterus_chaoyangensis",
# printZoom=FALSE)
```

17. `distNodes`

Title *Finding distance between nodes and tips*

Description

The function computes the distance between pairs of nodes, pairs of tips, or between nodes and tips. The distance is meant as both patristic distance and the number of nodes intervening between the pair.

Usage

```
distNodes(tree,node=NULL,clus=0.5)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

node either a single node/tip or a pair of nodes/tips.

clus the proportion of clusters to be used in parallel computing.

Value

If 'node' is specified, the function returns a data frame with distances between the focal node/tip and the other nodes/tips on the tree (or for the selected pair only). Otherwise, the

function returns a matrix containing the number of nodes intervening between each pair of nodes and tips.

Examples

```
### data("DataApes")

### DataApes$Tstage->Tstage


# distNodes(tree=Tstage)

# distNodes(tree=Tstage,node=64)

# distNodes(tree=Tstage,node="Tro_2")

# distNodes(tree=Tstage,n=c(64,48))

# distNodes(tree=Tstage,n=c(64, "Tro_2"))
```

18. `search.conv`

Title *Searching for morphological convergence among species and clades*

Description

The function scans a phylogenetic tree looking for morphological convergence between entire clades or species evolving under specific states.

Usage

```
search.conv(RR=NULL,tree=NULL,y,nodes=NULL,state=NULL,aceV=NULL,
```

```
min.dim=NULL,max.dim=NULL,min.dist=NULL,PGLSf=FALSE,declust=FALSE,nsim=1
000,rsim=1000,clus=0.5,foldername=NULL)
```

Arguments

RR an object produced by *RRphylo*. This is not indicated if convergence among states is tested.

tree a phylogenetic tree. The tree needs not to be ultrametric or fully dichotomous. This is not indicated if convergence among clades is tested.

y a multivariate phenotype. The object 'y' should be either a matrix or dataframe with species names as rownames.

nodes node pair to be tested. If unspecified, the function automatically searches for convergence among clades. Notice the node number must refer to the dichotomic version of the original tree, as produced by *RRphylo*.

state the state of the tips. If a single convergent state is indicated species within the state are tested. Otherwise, species belonging to different parts of the tree could be tested for convergence on each other by indicating different states. This latter case is especially meant to test for iterative evolution (i.e. the appearance of repeated morphotypes into different clades). The state for non-focal species (i.e. not belonging to any convergent group) must be indicated as "nostate". The state for non-focal species (i.e. not belonging to any convergent group) must be indicated as "nostate".

aceV phenotypic values at internal nodes. The object 'aceV' should be either a matrix or dataframe with nodes (referred to the dichotomic version of the original tree, as produced by *RRphylo*) as rownames. If 'aceV' are not indicated, ancestral phenotypes are estimated via *RRphylo*.

min.dim the minimum size of the clades to be compared. When node is indicated, it indicates the minimum size of the smallest clades in nodes, otherwise it is set at one tenth of the tree size.

max.dim the maximum size of the clades to be compared. When node is indicated, it is 'min.dim'*2 if the largest clade in 'nodes' is smaller than this value, otherwise it corresponds to the size of the largest clade. Without nodes it is set at one third of the tree size.

min.dist the minimum distance between the clades to be compared. When nodes is indicated, it is the distance between the pair. Under the automatic mode, the user can choose whether time distance or node distance (i.e. the number of nodes intervening between the pair) should be used. If time distance has to be considered, 'min.dist' should be a character argument containing the word "time" and then the actual time distance to be used. The same is true for node distance, but the word "node" must precede the node distance to be used. For example, if the user want to test only clades more distant than 10 time units, the argument should be "time10". If clades separated by more than 8 nodes has to be tested, the argument 'min.dist' should be "node8". If left unspecified, it automatically searches for convergence between clades separated by a number of nodes bigger than one tenth of the tree size.

PGLSf has been deprecated; please see the argument 'declust' instead.

declust if species under a given state (or a pair of states) to be tested for convergence are phylogenetically closer than expected by chance, trait similarity might depend on proximity rather than true convergence. In this case, by setting declust = TRUE, tips under the focal state (or states) are removed randomly until clustering disappears. A minimum of 3 species per state is enforced to remain anyway.

nsim number of simulations to perform sampling within the theta random distribution. It is set at 1000 by default.

rsim number of simulations to be performed to produce the random distribution of theta values. It is set at 1000 by default.

clus the proportion of clusters to be used in parallel computing.

foldername the path of the folder where plots are to be found.

Details

Regardless the case (either 'state' or 'clade'), the function stores a plot into the folder specified by 'foldername'. If convergence among clades is tested, the clade pair plotted corresponds to those clades with the smallest **$average distance from group centroids**. The figure shows the Euclidean distances computed between the MRCAs of the clades and the mean Euclidean distance computed between all the tips belonging to the converging clades, as compared to the distribution of these same figures across the rest of the tree. Furthermore, the function stores the PC1/PC2 plot obtained by PCA of the species phenotypes. Convergent clades are indicated by colored convex hulls. Large colored dots represent the mean phenotypes per clade (i.e. their group centroids). Eventually, a modified traitgram plot is produced, highlighting the branches of the clades found to converge. In both PCA and traitgram, asterisks represent the ancestral phenotypes of the individual clades. If convergence among states is tested, the function produces a PC plot with colored convex hulls enclosing species belonging to different states. Furthermore, it generates circular plots of the mean angle between states (blue lines) and the range of random angles (gray shaded area). The p-value for the convergence test is printed within the circular plots.

Value

If convergence between clades is tested, the function returns a list including:

• **$node pairs**: a dataframe containing for each pair of nodes:

– ang.bydist.tip: the mean theta angle between clades divided by the time distance.

– ang.conv: the mean theta angle between clades plus the angle between aces, divided by the time distance.

– ang.ace: the angle between aces.

– ang.tip: the mean theta angle between clades.

– nod.dist: the distance intervening between clades in terms of number of nodes.

– time.dist: the time distance intervening between the clades.

– p.ang.bydist: the p-value computed for ang.bydist.tip.

– p.ang.conv: the p-value computed for ang.conv.

– clade.size: the size of clades.

• **$node pairs comparison**: pairwise comparison between significantly convergent pairs (all pairs if no instance of significance was found) performed on the distance from group centroids (the mean phenotype per clade).

• **$average distance from group centroids**: smaller average distances mean less variable phenotypes within the pair.

If convergence between (or within a single state) states is tested, the function returns a dataframe including for each pair of states (or single state):

• $**ang.state**: the mean theta angle between species belonging to different states (or within a single state).

• **$ang.state.time**: the mean of theta angle between species belonging to different states (or within a single state) divided by time distance.

• **$p.ang.state**: the p-value computed for ang.state.

• **$p.ang.state.time**: the p-value computed for ang.state.time.

Examples

```
### data("DataFelids")

### DataFelids$PCscoresfel->PCscoresfel

### DataFelids$treefel->treefel

### DataFelids$statefel->statefel

# RRphylo(treefel,PCscoresfel)->RRfel


## Case 1. searching convergence between clades


## by setting min.dist as node distance

# search.conv(RR=RRfel, y=PCscoresfel, min.dim=5, min.dist="node9",

# foldername = tempdir())


## by setting min.dist as time distance

# search.conv(RR=RRfel, y=PCscoresfel, min.dim=5,

# min.dist="time38", foldername = tempdir())


## Case 2. searching convergence within a single state

# search.conv(tree=treefel, y=PCscoresfel, state=statefel,

# foldername = tempdir())
```

19. PGLS_fossil

Title *Phylogenetic Generalized Least Square with fossil phylogenies*

Description

The function performs pgls for non-ultrametric trees using either Pagel's lambda transform, Brownian Motion or *RRphylo* rates to change the correlation structure.

Usage

```
PGLS_fossil(modform,data,tree,RR=NULL)
```

Arguments

modform the formula for the regression model.

data a list of named vectors including response and predictor variables as named in 'modform'.

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

RR the result of *RRphylo* performed on the response variable. If NULL (default), the function fits Pagel's lambda in the regression for univariate data or uses the tree variance covariance matrix in the multivariate case. If 'RR' is specified, tree branches are rescaled to the absolute branch-wise rate values calculated by *RRphylo* to transform the variance-covariance matrix.

Value

Fitted pgls parameters and significance.

Details

With univariate data, the user may want to use either Pagel's lambda or *RRphylo* rates to transform the correlation structure. In the former case, the lambda transform is fitted to the data (Revell, 2010). In the latter case, branch lengths are multiplied by absolute rates as computed by *RRphylo* to accomodate rate variation across the tree. In the multivariate case, the variance-covariance structure is either left unaltered by keeping 'RR' = NULL (Adams and Collyer, 2015) or changed according to the norm-2 vector of rates computed for each phenotype by *RRphylo*.

Examples

```
# library(ape)
# library(phytools)
# rtree(100)->tree
# fastBM(tree)->y
# fastBM(tree)->x
# RRphylo(tree,y)->RR

# PGLS_fossil(y~x,RR=RR,type="Pagel")
# PGLS_fossil(y~x,RR=RR,type="RRphylo")

# library(ape)
# library(phytools)
# rtree(100)->tree
# fastBM(tree)->resp
# fastBM(tree,nsim=3)->resp.multi
```

```
# fastBM(tree)->pred1
# fastBM(tree)->pred2
# PGLS_fossil(modform=y1~x1+x2,data=list(y1=resp,x2=pred1,x1=pred2),
# tree=tree)
# RRphylo::RRphylo(tree,resp)->RR
# PGLS_fossil(modform=y1~x1+x2,data=list(y1=resp,x2=pred1,x1=pred2),
# tree=tree,RR=RR)
#
# PGLS_fossil(modform=y1~x1+x2,
# data=list(y1=resp.multi,x2=pred1,x1=pred2),tree=tree)
# RRphylo::RRphylo(tree,resp.multi)->RR
# PGLS_fossil(modform=y1~x1+x2,
# data=list(y1=resp.multi,x2=pred1,x1=pred2),tree=tree,RR=RR)
```

20. `StableTraitsR`

Title *Run StableTraits from within R*

Description

This function runs *StableTraits* and *StableTraitsSum* (Elliot and Mooers 2014) from within the R environment and returns its output into the workspace.

Usage

```
StableTraitsR(tree,y,path,output=NULL,aces=NULL,argST=NULL,argSTS=NULL)
```

Arguments

tree a phylogenetic tree. The tree needs not to be either ultrametric or fully dichotomous.

y a named vector of phenotypic trait.

path the folder path where the *StableTraits* output will be stored. Notice that the input tree and data (modified automatically if the original tree is not fully dichotomous or if aces are specified) will be stored in this folder as well.

output name of the output to be returned, if unspecified it will be named "output".

aces a named vector of ancestral character values at nodes specified in advance. Names correspond to the nodes in the tree.

argST a list of further arguments passed to *StableTraits*. If the argument has no value (for example "brownian") it must be specified as TRUE.

argSTS list of further arguments passed to *StableTraitsSum*. If the argument has no value (for example "brownian") it must be specified as TRUE.

Details

The StableTraits software is available at http://www.michaelelliot.net/stabletraits/, along with instructions for compilation. Once it is installed, the user must set as R working directory the folder where the *StableTraits* software are installed. Further information about the arguments and outputs of *StableTraits* and *StableTraitsSum* can be found at http://www.michaelelliot.net/stabletraits/. *StableTraitsR* automatically recognizes which Operating System is running on the computer (it has been tested successfully on MacOS and Windows machines).

Value

The function returns a list containing the output of *StableTraits* and *StableTraitsSum*.

$progress a table reporting the DIC and PRSF diagnostics.

$rates_tree a copy of the original tree with branch lengths set to the evolutionary rate imputed by the stable reconstruction. Specifically, each branch length is equal to the absolute difference in the stable reconstruction occurring on that branch divided by the square root of the input branch length.

$rates the original branch lengths, evolutionary rates, node height and (optionally) scaled branch lengths.

$aces the median estimates of ancestral states and stable parameters along with the 95% credible interval.

$brownian_tree if "brownian" is TRUE in 'argSTS', a copy of the original tree with branch lengths set such that the Brownian motion reconstruction of the character on this tree is approximately the same as the stable ancestral reconstruction.

$ace.prior.values if aces is specified, the function returns a dataframe containing the corresponding node number on the *RRphylo* tree for each node, the original (preset) and the estimated values, and the 95% credible interval.

Examples

```
# library(ape)
# library(phytools)
# setwd("~/Software/StableTraits")
# dir.create("Analyses")
# rtree(100)->tree
```

```
# fastBM(tree)->y
# c(1,2,3)->acev
# sample(Ntip(tree)+seq(1:Nnode(tree)),3)->names(acev)
# StableTraitsR(tree,y,path="Analyses/",output="my_output",aces=acev,
# argST=list(iterations=500000,chains=4),argSTS=list(brownian=TRUE))->ST
```

21. `overfitRR`

Title *Testing RRphylo methods overfit*

Description

Testing the robustness of *search.trend* (Castiglione et al. 2019a), *search.shift* (Castiglione et al. 2018), and *search.conv* (Castiglione et al. 2019b) to sampling effects and phylogenetic uncertainty.

Usage

```
overfitRR(RR,y,s=0.25,swap.args=NULL,trend.args=NULL,shift.args=NULL,
conv.args=NULL,aces=NULL,x1=NULL,aces.x1=NULL,cov=NULL,rootV=NULL,
nsim=100,clus=0.5)
```

Arguments

RR an object produced by *RRphylo*.

y a named vector of phenotypes.

s the percentage of tips to be cut off. It is set at 25% by default.

swap.args a list of arguments to be passed to the function *swapONE*, including 'list(si=NULL,si2=NULL,node=NULL)'. If 'swap.arg' is unspecified, the function automatically sets both 'si' and 'si2' to 0.1.

shift.args a list of arguments specific to the function *search.shift*, including 'list(node=NULL,state=NULL) '. Arguments node and state can be specified at the same time.

trend.args a list of arguments specific to the function *search.trend*, including 'list(node=NULL) '. If a trend for the whole tree is to be tested, type 'trend.args = list()'. No trend is tested if left unspecified.

conv.args a list of arguments specific to the function *search.conv*, including 'list(node=NULL,state=NULL,PGLS=FALSE)'. Arguments node and state can be specified at the same time.

aces if used to produce the 'RR' object, the vector of those ancestral character values at nodes known in advance must be specified. Names correspond to the nodes in the tree.

x1 the additional predictor to be specified if the 'RR' object has been created using an additional predictor (i.e. multiple version of *RRphylo*). 'x1' vector must be as long as the number of nodes plus the number of tips of the tree, which can be obtained by running *RRphylo* on the predictor as well, and taking the vector of ancestral states and tip values to form the 'x1'.

aces.x1 a named vector of ancestral character values at nodes for 'x1'. It must be indicated if the 'RR' object has been created using both aces and 'x1'. Names correspond to the nodes in the tree.

cov if used to produce the 'RR' object, the covariate must be specified. As in *RRphylo*, the covariate vector must be as long as the number of nodes plus the number of tips of

the tree, which can be obtained by running *RRphylo* on the covariate as well, and taking the vector of ancestral states and tip values to form the covariate.

rootV if used to produce the 'RR' object, the phenotypic value at the tree root must be specified.

nsim number of simulations to be performed. It is set at 100 by default.

clus the proportion of clusters to be used in parallel computing.

Details

Methods using a large number of parameters risk being overfit. This usually translates in poor fitting with data and trees other than the those originally used. With *RRphylo* methods this risk is usually very low. However, the user can assess how robust the results got by applying *search.shift* ,*search.trend,* or *search.conv* are by running *overfitRR*. With the latter, the original tree and data are subsampled by specifying a 's' parameter, that is the proportion of tips to be removed from the tree. Internally, *overfitRR* further shuffles the tree by using the function *swapONE*. Thereby, both the potential for overfit and phylogenetic uncertainty are accounted for straight away.

Value

The function returns a list containing:

$rootCI the 95% confidence interval around the root value.

$ace.regressions the results of linear regression between ancestral state estimates before and after the subsampling.

$conv.results a list including results for *search.conv* performed under clade and state conditions. If a node pair is specified within 'conv.args', the $clade object contains the

percentage of simulations producing significant p-values for convergence between the clades. If a state vector is supplied within 'conv.args', the object $state contains the percentage of simulations producing significant p-values for convergence within (single state) or between states (multiple states).

$shift.results a list including results for *search.shift* performed under “clade” and “sparse” conditions. If one or more nodes are specified within 'shift.args', the $clade object contains for each node the percentage of simulations producing significant p-value separated by shift sign, and the same figures by considering all the specified nodes as evolving under a single rate (all.clades).If a state vector is supplied within 'shift.args', the object $sparse contains the percentage of simulations producing significant p-value separated by shift sign ($p.states).

$trend.results a list including the percentage of simulations showing significant p-values for phenotypes versus age and absolute rates versus age regressions for the entire tree separated by slope sign ($tree). If one or more nodes are specified within 'trend.args', the list also includes the same results at nodes ($node) and the results for comparison between nodes ($comparison).

Examples

```
### Data("DataOrnithodirans")
### DataOrnithodirans$treedino->treedino
### DataOrnithodirans$massdino->massdino
### DataOrnithodirans$statedino->statedino

## Extract Pterosaurs tree and data
```

```
# library(ape)
# extract.clade(treedino,748)->treeptero
# massdino[match(treeptero$tip.label,names(massdino))]->massptero
# massptero[match(treeptero$tip.label,names(massptero))]->massptero
# RRphylo(tree=treedino,y=massdino)->dinoRates
# RRphylo(tree=treeptero,y=log(massptero))->RRptero

## Case 1. search.shift under both "clade" and "sparse" condition
# search.shift(RR=dinoRates, status.type= "clade",
# foldername=tempdir())->SSnode
# search.shift(RR=dinoRates, status.type= "sparse", state=statedino,
# foldername=tempdir())->SSstate

# overfitRR(RR=dinoRates,y=massdino,swap.args =list(si=0.2,si2=0.2),
# shift.args =list(node=rownames(SSnode$single.clades),
# state=statedino),nsim=10,clus=0.5)

## Case 2. search.trend on the entire tree
# search.trend(RR=RRptero, y=log(massptero),nsim=100, clus=0.5,
# foldername=tempdir(),cov=NULL,ConfInt=FALSE,node=NULL)->STtree

# overfitRR(RR=RRptero,y=log(massptero),swap.args =list(si=0.2,si2=0.2),
# trend.args = list(),nsim=10,clus=0.5)->overfit.STtree

## Case 3. search.trend at specified nodes
# search.trend(RR=RRptero,y=log(massptero),node=143,clus=0.5,
# foldername=tempdir(),cov=NULL,ConfInt=FALSE)->STnode
```

```
# overfitRR(RR=RRptero,y=log(massptero),trend.args=list(node=143),
# nsim=10,clus=0.5)

## Case 4. overfitRR on multiple RRphylo
### data("DataCetaceans")
### DataCetaceans$treecet->treecet
### DataCetaceans$masscet->masscet
### DataCetaceans$brainmasscet->brainmasscet
### DataCetaceans$aceMyst->aceMyst
# ape::drop.tip(treecet,treecet$tip.label[-match(names(brainmasscet),
# treecet$tip.label)])->treecet.multi
# masscet[match(treecet.multi$tip.label,names(masscet))]->masscet.multi
# RRphylo(tree=treecet.multi,y=masscet.multi)->RRmass.multi
# RRmass.multi$aces[,1]->acemass.multi
# c(acemass.multi,masscet.multi)->x1.mass
# RRphylo(tree=treecet.multi,y=brainmasscet,x1=x1.mass)->RRmulti
# search.trend(RR=RRmulti, y=brainmasscet,x1=x1.mass,clus=0.5,
# foldername=tempdir())->STcet

# overfitRR(RR=RRmulti,y=brainmasscet,trend.args = list(),
# x1=x1.mass,nsim=10,clus=0.5)

## Case 5. searching convergence between clades and within a single state
### data("DataFelids")
### DataFelids$PCscoresfel->PCscoresfel
### DataFelids$treefel->treefel
```

```
### DataFelids$statefel->statefel
# RRphylo(tree=treefel,y=PCscoresfel)->RRfel
# search.conv(RR=RRfel, y=PCscoresfel, min.dim=5, min.dist="node9",
# foldername = tempdir())->SC.clade
# as.numeric(c(rownames(SC.clade[[1]])[1],
# as.numeric(as.character(SC.clade[[1]][1,1]))))->conv.nodes

# overfitRR(RR=RRfel, y=PCscoresfel,conv.args = list(node=conv.nodes,
# state=statefel,declust=TRUE),nsim=10,clus=0.5)
```

22. `scaleTree`

Title *Phylogenetic tree calibration*

Description

The function is a wrapper around the functions "scalePhylo", "assign.ages", and "assign.brlen" written by Gene Hunt (http://paleobiology.si.edu/staff/individuals/hunt.cfm) . It rescales tree branch lengths according to given calibration dates.

Usage

```
scaleTree(tree,tip.ages,node.ages=NULL,min.branch=0.1)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

tip.ages a named vector including the ages (i.e. distance from the youngest tip within the tree) of the tips to be changed. If unspecified, the function assumes all the tips are correctly placed with respect to the root.

node.ages a named vector including the ages (i.e. distance from the youngest tip within the tree) of the nodes to be changed. If no calibration date for nodes is supplied, the function shifts node position only where needed to fit tip ages.

min.branch the minimum branch length that will be imposed for shifted nodes.

Value

Rescaled phylogentic tree.

Examples

```
# library(ape)
# library(phytools)
# library(geiger)
### data("DataFelids")
### DataFelids$treefel->tree
# max(nodeHeights(tree))->H

#### Example 1 ####
# rep(0,4)->tipAges
# names(tipAges)<-tips(tree,146)
# scaleTree(tree,tipAges)->treeS1

# edge.col<-rep("black",nrow(tree$edge))
```

```
# edge.col[which(treeS1$edge[,2]%in%getDescendants(treeS1,146))]<-"red"
# layout(2:1)
# plot(tree,edge.color = edge.col,show.tip.label=F)
# plot(treeS1,edge.color = edge.col,show.tip.label=F)

#### Example 2 ####
# nodeAges<-c(23.5,15.6)
# names(nodeAges)<-c(85,139)

# scaleTree(tree,node.ages=nodeAges)->treeS2

# edge.col<-rep("black",nrow(tree$edge))
# edge.col[which(treeS1$edge[,2]%in%c(getDescendants(treeS1,85),
# getDescendants(treeS1,139)))]<-"red"
# layout(2:1)
# plot(tree,edge.color = edge.col,show.tip.label=F)
# nodelabels(bg="w",frame="n",node=c(85,139),col="green")
# plot(treeS2,edge.color = edge.col,show.tip.label=F)
# nodelabels(bg="w",frame="n",node=c(85,139),col="green")

#### Example 3 ####
# 16->nodeAges
# names(nodeAges)<-"145"
# tipAges<-19
# names(tipAges)<-tree$tip.label[1]

# scaleTree(tree,tip.ages = tipAges,node.ages=nodeAges)->treeS3
```

```
# edge.col<-rep("black",nrow(tree$edge))
# edge.col[which(treeS3$edge[,2]%in%c(1,getMommy(tree,1),
# getDescendants(treeS3,145)))]<-"red"
# layout(2:1)
# plot(tree,edge.color = edge.col,show.tip.label=F)
# nodelabels(bg="w",frame="n",node=145,col="green")
# plot(treeS3,edge.color = edge.col,show.tip.label=F)
# nodelabels(bg="w",frame="n",node=145,col="green")
```

23.phyloclust

Title *Test for phylogenetic clustering*

Description

The function tests the presence of phylogenetic clustering for species within a focal state.

Usage

```
phyloclust(tree,state,focal,nsim=100)
```

Arguments

tree a phylogenetic tree. The tree needs not to be ultrametric and fully dichotomous.

state the named vector of tip states.

focal the focal state to be tested for phylogenetic clustering.

nsim number of simulations to perform the phylogenetic clustering test.

Details

To test for phylogenetic clustering, the function computes the mean cophenetic (i.e. evolutionary time) distance between all the species under the 'focal' state. Such value is compared to a random distribution of time distances obtained by sampling 'nsim' times as many random tips as those under the 'focal' state. In the presence of significant phylogenetic clustering, tips under the 'focal' state are randomly removed until the p becomes >0.05 or only 3 tips are left.

Value

The function returns a list including the p-value ($p) for the test of phylogenetic clustering and a $declusterized object containing the declusterized versions of the original tree and state vector (i.e. tips are removed as to make p>0.05) and the vector of removed species.

Examples

```
### data("DataFelids")

### DataFelids$treefel->treefel

### DataFelids$statefel->statefel

# phyloclust(tree=treefel,state=statefel,focal="saber")
```

References

Adams, D.C., & Collyer, M. L. (2017). Multivariate phylogenetic comparative methods: evaluations, comparisons, and recommendations. Systematic Biology, **67**, 14-31. https://doi.org/10.1093/sysbio/syx055

Castiglione, S., Tesone, G., Piccolo, M., Melchionna, M., Mondanaro, A., Serio, C., Di Febbraro, M., & Raia, P. (2018). A new method for testing evolutionary rate variation and shifts in phenotypic evolution. *Methods in Ecology and Evolution*, in press.doi:10.1111/2041-210X.12954

Castiglione, S., Serio, C., Mondanaro, A., Di Febbraro, M., Profico, A., Girardi, G., & Raia, P. (2019a). Simultaneous detection of macroevolutionary patterns in phenotypic means and rate of change with and within phylogenetic trees including extinct species. *PLoS ONE*, 14: e0210101. https://doi.org/10.1371/journal.pone.0210101

Castiglione, S., Serio, C., Tamagnini, D., Melchionna, M., Mondanaro, A., Di Febbraro, M., Profico, A., Piras, P.,Barattolo, F., & Raia, P. (2019b). A new, fast method to search for morphological convergence with shape data. *PLoS ONE*, 14, e0226949. https://doi.org/10.1371/journal.pone.0226949

Clavel, J., Escarguel, G., & Merceron, G.(2015). mvMORPH: an R package for fitting multivariate evolutionary models to morphometric data. Methods in Ecology and Evolution, 6:1311–1319. doi: 10.1111/2041-210X.12420

Elliot, M. G., & Mooers, A. Ø. (2014). Inferring ancestral states without assuming neutrality or gradualism using a stable model of continuous character evolution. BMC evolutionary biology, 14: 226. doi.org/10.1186/s12862-014-0226-8

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

O'Meara, B. C., Ané, C., Sanderson, M. J., & Wainwright, P. C. (2006). Testing for different rates of continuous trait evolution using likelihood. *Evolution*, **60**, 922-933.

Piras, P., Silvestro, D., Carotenuto, F., Castiglione, S., Kotsakis, A., Maiorino, L., Melchionna, M., Mondanaro, A., Sansalone, G., Serio, C., Vero, V.A., Raia, P. (2018) Evolution of the sabertooth mandible: A deadly ecomorphological specialization. Palaeogeography, Palaeoclimatology, Palaeoecology, in press.

Profico, A., Piras, P., Buzi, C., Di Vincenzo, F., Lattarini, F., Melchionna, M., Veneziano, A., Raia, P. & Manzi, G. (2017). The evolution of cranial base and face in Cercopithecoidea and Hominoidea: Modularity and morphological integration. American journal of primatology, 79(12).

Raia, P., Carotenuto, F., Meloro, C., Piras, P., & Pushkina, D. (2010). The shape of contention: adaptation, history, and contingency in ungulate mandibles. Evolution, 64(5), 1489-1503.

Revell, L.J. (2010). Phylogenetic signal and linear regression on species data. *Methods in Ecology and Evolution*, **1**, 319-329. https://doi.org/10.1111/j.2041-210X.2010.00044.x

Revell, L. J. (2012) phytools: An R package for phylogenetic comparative biology (and other things). *Methods in Ecology and Evolution*, **3,** 217-223.<doi:10.1111/j.2041-210X.2011.00169.x>

Suzuki, R., & Shimodaira, H. (2015). pvclust: Hierarchical Clustering with P-Values via Multiscale Bootstrap Resampling. R package version 2.0-0. https://CRAN.R-project.org/package=pvclust