Calculation of evolutionary rates and rate shifts with the RRphylo method

Contents

14.plotRates function to plot RRphylo rates at a specified node

15.search.trend searching for evolutionary trends in phenotypes and rates

Depends ape, phytools, geiger, stats4, lmtest, scales, parallel, pbapply, bbmle, R.utils, smatr

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)
```

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.

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

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

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 without accounting for the effect of a covariate
## performing RRphylo on the covariate (body mass itself) 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)
```

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, 0.001 units apart, ranging from -10 to 10 (please notice that since version 1.1 the estimation of λ is restricted in between 0 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
setBM(tree,1,type="brown")->y

## perform RRphylo
RRphylo(tree,y)->RR
RR$tip.path->L
RR$node.path->L1

## compute the phylogenetic signal at different values of lambda, taken
from a uniform distribution of lambda values
seq(-10,10,.001)->lambdaD
lambdaD[-10001]->lambdaD
f=function(lambda){
require(phytools)
```

```
betas <- (solve(t(L) %*% L + lambda * diag(ncol(L))) %*% t(L)) %*%
as.matrix(y)
aceRR <- L1 %*% betas[1:Nnode(tree), ]
y.hat <- L %*% betas
return(phylosig(tree,y.hat,test=TRUE))
}
library(parallel)
library(pbapply)
cl<-makeCluster(detectCores()-1)
clusterExport(cl, c("f", "L", "L1", "y", "tree"))
pbsapply(lambdaD,f,cl=cl)->Ks
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"
col[which(Ps>.05)]<-"green"
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)
```

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 . Yet, the phylogenetic signal becomes non-significant (i.e. phylogenetic effects disappear) for even smaller absolute values of λ (the green dots in Figure 1). 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.

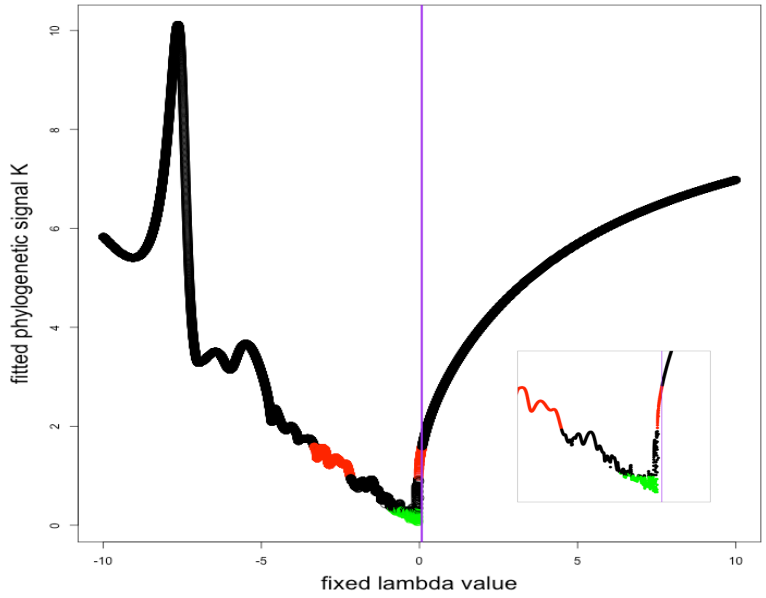

Figure 1 The phylogenetic signal K at different values of the penalization factor λ. The initial phenotype was simulated according to the Brownian motion model of evolution. As expected, the fitted λ value (indicated by the purple vertical line) produces a phylogenetic signal close to 1. The

small box on the right represents the same plot as in the larger figure, yet cut down to the region of small λ values. Please notice that since version 1.1 the estimation of λ is restricted in between 0 to 10

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.

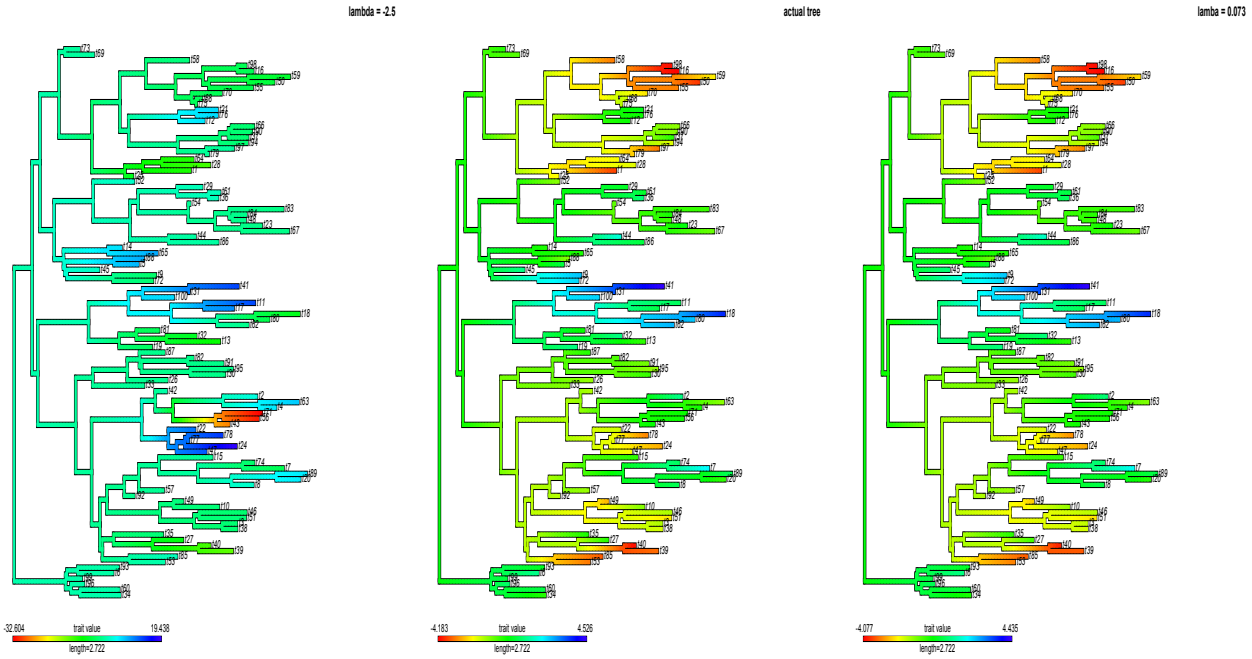

Figure 2 The distribution of phenotypes produced with λ values giving a K value close to the original, Brownian motion simulation. The original phenotypes are reported in the tree at the center of the figure. The phenotypes with the λ value fitted in RRphylo are to the right. The tree to the left represent 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 2 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)
  aceRR <- L1 %*% betas[1:Nnode(t), ]
  y.hat <- L %*% betas
  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)
```

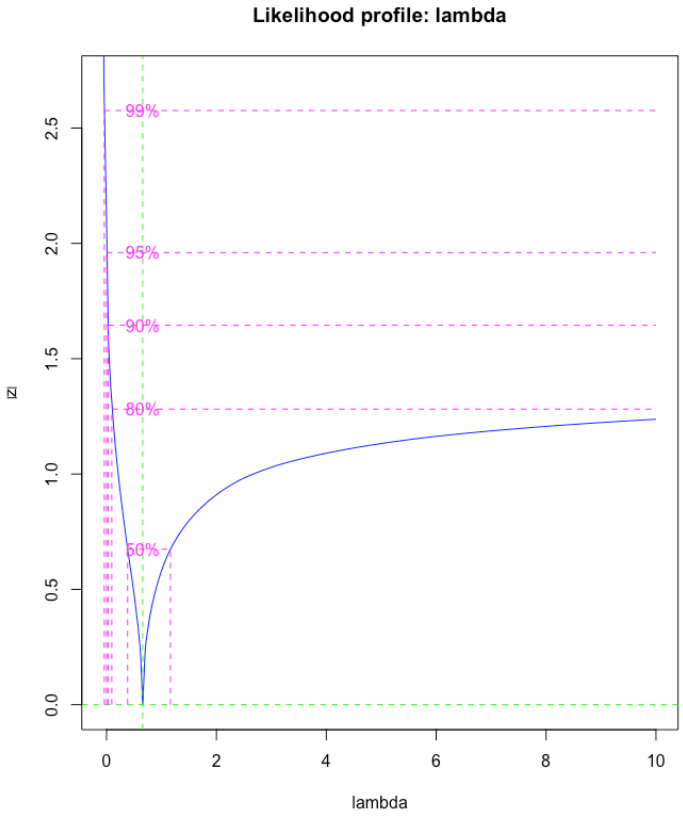

Figure 3 The likelihood profile of the *optL* function, used within *RRphylo* for the purpose of penalization.

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"),
auto.recognize = c("yes","no"), node = NULL, test.single =
c("yes", "no"), 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**.**

auto.recognize indicates whether individual clades must be tested for rate shift without a priori indication.

node under the "clade" condition, the node of the tree to be tested for the rate shift. When multiple nodes are tested, they need to be written as in the example below.

test.single whether multiple nodes indicated under the "clade" condition should be tested individually ('yes') or not ('no').

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 (Castiglione et al. 2018) takes the object produced by RRphylo. Two different conditions of rate change can be investigated. Under the 'clade' conditions 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 auto.recognize, a list including:

$`all different clades` the probability that the nodes subtending to the largest and smallest average rates (in absolute values) do represent a real shift. Probabilities are contrasted to simulations shuffling the rates across the tree branches, a number of

times 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 $p < 0.025$.

$`all clades rate difference` the nodes subtending to the largest and smallest rates (in absolute values) in the tree, and the average rate differences, computed as the mean rate over all branches subtended by the node, minus the average rate for the rest of the tree.

$`shift test single clades` the same as with 'all different 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 $p < 0.025$.

$`average rate difference` the average rate differences for the branches descending by the nodes indicated in `shift test single clades`

Clade condition, without auto.recognize:

$`shift test` this specifies the significance of the rate shift test, by considering all the specified nodes as a 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.

$`shift test single clades` if test.single=TRUE, this gives the significance for individual clades, tested separately. 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.

$`average rate difference` this is 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. Nodes along a single path are not permitted. In this case, only the largest/smallest rate values are selected 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:

$`rate difference` this is the average rate difference, computed as the mean rate over all leaves (terminal nodes) evolving under a given state, minus the average rate for each other state.

$`p rate diff` the probability that the shift under the sparse condition is real. 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. States are compare pairwise.

Examples

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

## RRphylo(tree=treedino,y=massdino)->dinoRates

## Case 1. Without accounting fo the effect of a covariate

## Case 1.1 "clade" condition
## with auto.recognize
#search.shift(dinoRates,auto.recognize="yes",test.single= "no",
status.type= "clade")
## testing two hypothetical clades
# search.shift(dinoRates,auto.recognize= "no",test.single= "yes",
status.type= "clade", node=c(node1,node2))

## Case 1.2 "sparse" condition
## testing the sparse condition. It refers to unrelated species, that
undergo phenotypic evolution under the same state (e.g.locomotory type,
environmental regime, constraint etc. etc.)
```

```
# search.shift(dinoRates,auto.recognize="no",test.single="no",
status.type="sparse", state=statedino)


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


## Case 2.1 "clade" condition
# search.shift(dinoRates,auto.recognize="yes",test.single="no",
status.type="clade", cov=massdino)


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

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 = 5, 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 mean or in the rate of evolution (actually the residuals of the phenotype versus time regression, established via Breusch-Pagan test). A variation in the phenotypic mean over time (a phenotypic trend) is obtained by selection the option "drift". A trend in the rate of evolution should produce an increased variance in the residuals over time, which can be inspected under the option "trend", by means of Breusch-Pagan test for residuals heteroscedasticity.

trend.type two kinds of heteroscedastic residuals are generated under the "trend" type. The option "linear" produces a linear increase in heteroscedasticity, whereas the "stepwise" option produces an increase after a specified point in time.

tr the intensity of the trend in homoscedasticity 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 increase 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* = 0 the phenotype will be trendless, with *es* < 0 the variance will decrease exponentially towards the present and the other way around with *es* > 0.

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 either in the phenotypes themselves, or in the residuals of the phenotype versus age regression. The user may specify whether she wants trendless data (option "brown"), phenotypes trending in time (option "drift"), or phenotypes whose variance increases/decreases 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 change in variance (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 $1 * 10^{-3}$.

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. Find a node subtending to a clade of desired size

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 minimu 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.be
tween.species"),random=c("yes","no"),focus=c("node","species","both","no
ne"),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

Irrespective of the cases, *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 randomizatin

# 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

Thi 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=NULL,y=NULL,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.swap.one

Title *create alternative phylogenies from a given tree*

Usage

```
swapONE (tree,si=0.5,si2=0.5)
```

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.

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.

Details

swap.one 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


# swapONE(tree = treedino)
```

14. `plotRates`

Title *plot RRphylo rates at a specified node*

Usage

```
plotRates(RR,node,export.tiff = c(TRUE,FALSE) ,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 "yes" the function save a "rate_bars.tiff" file inside the working directory.

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,node=NULL,nsim=100,clus=.5,foldername,cov=NULL,
ConfInt=c(FALSE,TRUE))
```

Arguments

RR an object produced by RRphylo.

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

node number of nodes to be specifically tested for trends. It is NULL by default. Notice the node number must refer to the dichotomic version of the original tree, as produced by RRphylo.

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**.**

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

cov the covariate values to be specified if the RR object has been created with covariate.

ConfInt if TRUE, the function returns 95% confidence intervals for slopes of phenotype versus age, absolute rates versus age, and relative rates versus age regressions.

Details

The function simultaneously returns the regression of phenotypes and rates against age (in terms of distance from the tree root). As an output, plots for the absolute and relative rates regressions versus age are saved as a single ".pdf" file. In the plots, the 95% confidence

intervals of simulated phenotypes and rates for each node are plotted as shaded areas. The same is performed for the phenotype versus age regression.

Slopes and significance are reported for all regressions. In addition, slopes are compared to a family of simulated slopes (where the number of simulations is equal to *nsim*), produced as to show no phenotypic or rate trend, using the *setBM* function. In the simulations, the phenotypic value at the root and the Brownian rate are the same as with the original data. The penalization factor λ is fitted around the real λ value in each simulation. Eventually, a global p-value is provided. In the case of "drift" such value equals to the least significant between p.real and p.random. In the case of "trend" this corresponds to p.random. We empirically found these are the best approach to minimize Type I and Type II error rates.

In the case of "trend" regression, the function throws a warning if the tips are concentrated towards the present, which makes Type II error rate higher, especially for small trees and shallow slopes.

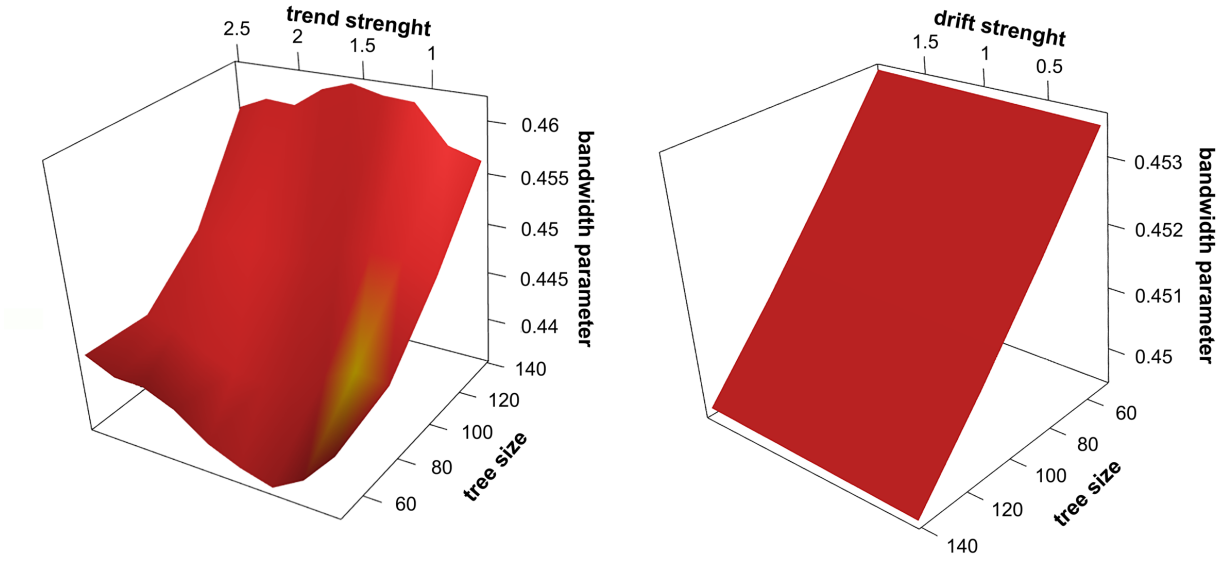

Figure 4. The power of search.trend to find a real pattern in either the evolutionary rate (left) or the phenotypic mean (right) as a function of tree size, intensity of the pattern, and

the distribution of tip ages. The red portion of the power surface represents the domain of "true positives" (real patterns that are in fact found to be significant).

Value

The function returns a 'list' object including:

$rbt: for each branch of the tree, there are the age of the daughter node/tip (D.age), the age of the parental node (P.age), the RRphylo rates, and the distance from the tree root (age). If y is multivariate, it also includes the multiple rates for each y component.

$pbt: a data frame of phenotypic values and distance from the tree root for each node and tip.

$p.trend: results of phenotype versus age regression. It reports a p-value for the regression between the variables (p.real), a p-value for the difference from 0 under standard major axis regression (p.sma0), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), a global p-value corresponding to the least significant between p.real and p.random (p.value; see details for further explanations).

$rbt.rateA: results of absolute rate values versus age regression. It reports a p-value for the regression between the variables (p.real), a p-value for the difference from 0 under standard major axis regression (p.sma0), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), and a global p-value (p.value, corresponding to p.random; see details for further explanations).

$rbt.rateR: results of rate values versus age regression. It reports a p-value for the regression between the variables (p.real), a p-value for the difference from 0 under standard major axis regression (p.sma0), a p-value computed contrasting the real slope to Brownian motion simulations (p.random), and a global p-value (p.value, corresponding to p.random; see details for further explanations).

$ConfInts: the 95% confidence intervals around the slopes of phenotype versus age ($phenotype), absolute rates versus age ($abs.rate), and relative rates versus age ($rel.rate) regressions.

If the node argument is specified, the list also includes **$p.trend.nodes**, **$rbt.rateA.nodes**, **$rbt.rateR.nodes**, which return the same results as above for each specified node. Finally, the **$SMA** object contains the comparisons between slopes of regression lines of each pair of nodes, for all the regressions previously performed, under standard major axis regression.

Examples

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

### Extract the Pterosaurs tree and data
extract.clade(treedino,748)->treeptero
assdino[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 specifically
tested for trends
```

```
# search.trend(RR=RRptero, y=log(massptero), nsim=100, clus=0.5,
foldername=tempdir(),cov=NULL,ConfInt=FALSE)

## 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 specifically
tested for trends
# search.trend(RR=RRpteroCov, y=log(massptero), nsim=100, clus=0.5,
foldername=tempdir(),cov=NULL,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(),cov=NULL,ConfInt=FALSE,cov=cov.values)
```

References

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

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.

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>

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