Package: geiger (via r-universe)

October 15, 2026

Type Package

Title Analysis of Evolutionary Diversification

Version 2.0.12

Date 2026-08-31

Depends ape (>= 3.0-6), R (>= 2.15.0), phytools (>= 1.5-1)

Imports MASS, mvtnorm, subplex, deSolve (>= 1.7), digest, Rcpp (>= 0.11.0), coda, ncbit, colorspace, methods

Suggests TreeSim

LinkingTo Rcpp

Description Methods for fitting macroevolutionary models to phylogenetic trees Pennell (2014) <doi:10.1093/bioinformatics/btu181>.

URL https://github.com/mwpennell/geiger-v2

License GPL (>= 2)

ByteCompile yes

NeedsCompilation yes

Author Luke Harmon [aut, cre], Matthew Pennell [aut], Chad Brock [aut], Joseph Brown [aut], Wendell Challenger [aut], Jon Eastman [aut], Rich FitzJohn [aut], Rich Glor [aut], Gene Hunt [aut], Liam Revell [aut], Graham Slater [aut], Josef Uyeda [aut], Jason Weir [aut], CRAN team [ctb] (corrections in 2022)

Maintainer Luke Harmon <lukeh@uidaho.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-08-31 22:30:08 UTC

RemoteUrl https://github.com/cran/geiger

RemoteRef HEAD

RemoteSha 59005d9c2d4477013922d4773333ed341b287ad3

Contents

geiger-package . . . 3
aicm . . . 3
aicw . . . 4
aov.phylo . . . 5
bd.ms . . . 6
calibrate.mecca . . . 8
calibrate.rjmcmc . . . 10
congruify.phylo . . . 11
dcount . . . 12
drop.extinct . . . 13
dtt . . . 14
fitContinuous . . . 16
fitContinuousMCMC . . . 21
fitDiscrete . . . 24
gbresolve . . . 27
geiger-data . . . 29
geiger-defunct . . . 31
geiger-example . . . 32
load.rjmcmc . . . 32
make.gbm . . . 34
mecca . . . 36
medusa . . . 39
name.check . . . 42
nh.test . . . 43
nodelabel.phylo . . . 44
plot.medusa . . . 46
pp.mcmc . . . 48
r8s.phylo . . . 51
ratematrix . . . 52
rc . . . 53
rescale.phylo . . . 55
rjmcmc.bm . . . 57
sim.bd . . . 60
sim.bdtree . . . 61
sim.char . . . 63
startingpt.mecca . . . 64
subset.phylo . . . 66
tips . . . 66
to.auteur . . . 67
treedata . . . 68

Index **70**

geiger-package *GEIGER*

Description

A package for macroevolutionary simulation and estimating parameters related to diversification from comparative phylogenetic data.

Details

Package:	geiger
Type:	Package
License:	GPL version 2 or greater?

Author(s)

LJ Harmon, J Weir, C Brock, RE Glor, W Challenger, G Hunt, R FitzJohn, MW Pennell, GJ Slater, JW Brown, J Uyeda, and JM Eastman

Maintainer: Matt Pennell <mwpennell@gmail.com>

aicm *Akaike's Information Criterion for MCMC samples (AICM)*

Description

Computes Akaike's Information Criterion for MCMC samples (AICM: see Raftery et al. 2007). Can be used to perform model selection using output from fitContinuousMCMC.

Usage

```
aicm(x)
```

Arguments

x	a vector containing sampled likelihoods from the MCMC chain. Assumes that burn-in has been removed prior to computation of AICM score.

Details

AICM is one way of comparing model fit using posterior likelihood samples. It has advantages over approaches such as thermodynamic integration in that it uses the chain output directly and thus has little added time cost. Furthermore, it has been shown to perform better than the harmonic mean estimator of the marginal likelihood. However, it is also less robust than stepping-stone or thermodynamic integration approaches and should be used with care.

Value

AICM - Akaike's Information Criterion for the posterior sample

Author(s)

Graham Slater

References

Raftery et al. 2007. Estimating the integrated likelihood via posterior simulation using the harmonic mean identity. In. Bernardo et al. (eds) Bayesian Statistics. Oxford University Press.

Examples

```
## generate a random set of values from a normal distribution,
## similar to a set of likelihood samples obtained via MCMC.

x <- rnorm(1000, -275, 2);
aicm(x);
```

`aicw` *determining Akaike weights*

Description

This function computes Akaike Weights and ranks model based on their support from a vector of AIC scores.

Usage

```
aicw(x)
```

Arguments

`x`	a named vector of AIC scores

Value

An nx3 matrix, where n is the number of models being compared. The first column contains the AIC scores, the second contains the deltaAIC score and the third the Akaike Weight. Models are ranked in descending order according to weight. Rownames are the model names taken from the input vector.

Author(s)

Graham Slater

Examples

```
AIC.scores <- c(3,7,-5, 6)
names(AIC.scores) <- c("model1", "model2", "model3", "model4")
aicw(AIC.scores)
```

`aov.phylo` *phylogenetic ANOVA and MANOVA*

Description

computing phylogenetic ANOVA or MANOVA

Usage

```
aov.phylo(formula, phy, nsim = 1000,
    test = c("Wilks", "Pillai", "Hotelling-Lawley", "Roy"), ...)
```

Arguments

`formula`	a formula specifying the model (see **Examples**)
`phy`	a phylogenetic tree of class 'phylo'
`nsim`	number of simulations to run
`test`	test statistic to apply if MANOVA
`...`	additional arguments to be passed to `print.anova`

Details

This function performs an ANOVA or MANOVA in a phylogenetic context. First, the test statistic for ANOVA (one dependent variable) or MANOVA (more than one dependent variable) is calculated. The null distribution of this test statistic is then obtained by simulating new sets of dependent variables on the phylogenetic tree. Simulations are run under a Brownian-motion model. For ANOVA, the rate parameter is estimated from the average squared independent contrast; for MANOVA the simulations use an estimated variance-covariance matrix from `ratematrix`.

For MANOVA, you can specify the test statistic for the summary table. Wilks' statistic is most popular in the literature and for further details, see `summary.manova`.

Value

The function prints (and returns) a standard ANOVA or MANOVA table and p-value based on simulations (from the `Pr(phy)` column). For convenience, the summary table is included as an attribute of the returned object (see **Examples**).

Author(s)

JM Eastman and LJ Harmon

References

Garland T Jr, AW Dickerman, CM Janis, and JA Jones. 1993. Phylogenetic analysis of covariance by computer simulation. *Systematic Biology* 42(3):265-292.

See Also

aov; anova; summary.manova; ratematrix

Examples

```
## Not run:
geo=get(data(geospiza))
dat=geo$dat
d1=dat[,1]
grp<-as.factor(c(rep(0, 7), rep(1, 6)))
names(grp)=rownames(dat)

## MANOVA
x=aov.phylo(dat~grp, geo$phy, nsim=50, test="Wilks")
print(attributes(x)$summary) # summary table

## ANOVA
x1=aov.phylo(d1~grp, geo$phy, nsim=50)

## End(Not run)
```

---

bd.ms *estimate net diversification rate*

Description

estimating net diversification rate with confidence limits and testing diversities

Usage

```
bd.ms(phy=NULL, time, n, missing = 0, crown=TRUE, epsilon = 0)
bd.km(phy=NULL, time, n, missing = 0, crown=TRUE)
crown.p(phy=NULL, time, n, r, epsilon)
stem.p(phy=NULL, time, n, r, epsilon)
crown.limits(time, r, epsilon, CI=0.95)
stem.limits(time, r, epsilon, CI=0.95)
```

Arguments

phy	a phylogenetic tree of class 'phylo' (see **Details**)
time	time interval (can be a vector)
n	number of extant species

r	net diversification rate, birth - death
epsilon	extinction rate as a fraction of speciation rate
missing	number of taxa missing from tree
crown	whether time is treated as crown age (or otherwise as stem age)
CI	confidence level for estimated parameters

Details

bd.ms uses the Magallon and Sanderson (2000) method to calculate net diversification rate for a clade given extant diversity and age. bd.km computes the Kendall-Moran estimate of speciation rate, which assumes a complete phylogenetic tree.

Associated functions crown.p and stem.p also calculate the probability of obtaining a clade with at least n species given a net diversification rate (r), extinction fraction (epsilon), and time interval. Associated functions stem.limits and crown.limits generate confidence limits on extant diversity given a net diversification rate (r), extinction fraction (epsilon), and time interval.

Where a function calls for a time and an n element, a tree may be given instead (as argument phy). The argument n is taken from the number of tips in the tree. The method will attempt to discern whether the model should be fitted assuming crown or stem. If the tree has a non-NULL phy$root.edge element, the length will be assumed for the stem and crown is assumed to be FALSE (see also read.tree).

Value

bd.ms returns net diversification rate (r = lambda - mu)

bd.km returns speciation rate assuming a completely sampled tree

crown.p and stem.p return the probability of obtaining a clade as big as (or bigger than) size n, given time, r, and epsilon

crown.limits and stem.limits return lower (lb) and upper (ub) confidence intervals for clade size given time, r, and epsilon

Author(s)

LJ Harmon and C Brock

References

Magallon S and MJ Sanderson. 2000. Absolute diversification rates in angiosperm clades. *Evolution* 55:1762-1780.

Examples

```
geo=get(data(geospiza))

# Assuming no extinction
bd.ms(phy=geo$phy, missing=1, epsilon=0)
```

```
# Assuming high extinction
bd.ms(phy=geo$phy, missing=1, epsilon=0.9)
```

`calibrate.mecca` *calibrating MECCA*

Description

Runs a user defined number of calibration simulations (Wegmann et al. 2009) to generate tuning parameters MECCA's ABC-MCMC. The function takes a large number of arguments but in most cases these can remain at default settings.

Usage

```
calibrate.mecca(phy, richness, model = c("BM", "Trend", "twoRate"),
 prior.list = list(priorSigma = c(-4.961845, 4.247066), priorMean = c(-10, 10)),
 Ncalibrations = 10000, sigmaPriorType = "uniform", rootPriorType = "uniform",
 divSampleFreq = 0, initPars = "ML", propWidth = 0.1,
        SigmaBounds = c(-4.961845, 4.247066),
 hotclade = NULL, BOXCOX = TRUE, optimRange = c(-1000, 10))
```

Arguments

`phy`	A time calibrated phylogenetic tree of class "phylo"
`richness`	A named vector of species richnesses corresponding to tips in the tree; if a tip does not represent a higher level taxon, its richness should be 1.
`model`	The model of trait evolution to be used. Options currently implemented are: "BM" = Brownian Motion, "Trend" = Brownian moton with a trend, "twoRate" = two Brownian rate model (see hotclade below).
`prior.list`	A namedlist containing prior distribution parameters. If no values are specified, default values will be used. Default values for the BM rate are taken from the range reported in Harmon et al. 2010.
`Ncalibrations`	The number of calibration simulations to use. No fewer than 10,000 are recommended.
`sigmaPriorType`	The type of prior distribution on the Brownian rate parameter. Currently, uniform and normal are implemented.
`rootPriorType`	The type of prior distribution on the root state rate parameter. Currently, uniform is implemented.
`divSampleFreq`	Should new trees be simulated at every step? The default (0) is yes. If a non-zero value is given, this will determine the frequency (every n steps) with which new tip trees are simulated. This option is designed for use with large datasets where simulating new trees will slow MECCA down considerably.

`initPars` Should Maximum Likelihood ("ML") values be used to start the diversification rate sampler? if not, two numeric values should be specified for speciation and extinction rates.

`propWidth` Proposal width for the diversification MCMC. The default value of 0.1 seems to work well.

`SigmaBounds` If a normal prior is used for the trait evolution rate, it can be bounded with reasonable values to ensure that the chain does not get stuck in areas of low likelihood. Default values correspond to the range of rates reported in Harmon et al. (2010).

`hotclade` If a two-rate model is to be fit, which clade takes the second rate? There are two ways to do this - either a clade is designated or a single edge. For a clade, two tip names that span the clade of interest should be specified in a vector. If a single edge is to be used pass a two-item vector, where the first item is a single terminal name (for a tip branch) or the number of a node (for an internal edge) and the second item is NULL

`BOXCOX` Will summaries be BOX-COX standardized? Default is yes and is recommended. Raw summaries will be returned, along with parameters for Box-Cox transformation.

`optimRange` Optimization range for the lambda parameter for Box-Cox. Default params should work in most cases.

Value

`$diversification`
A 3-column matrix containing 1) birth-rates and 2) death-rates sampled by the MCMC, plus 3) their associated likelihoods

`$trait` A matrix of trait evolution parameters sampled from their priors and summary statistics from the simulations. If BM is used, columns 1 and 2 are the rate and root state parameter samples. If Trend or twoRate are the models, the first two columns are the same but the third contains the additional rate or trend parameter

`$stdz`, `$lambda`, `$GM`, `$BoxCox`
If BOXCOX = T, these items will also be output to BoxCox transform simulations in the MCMC

Note

Tree simulation is done using modified code from Tanja Stadler's TreeSim package. Trait evolution is done using modified code from Liam Revell's phytools package.

Author(s)

Graham J Slater, Luke Harmon, Dan Wegmann

References

Slater, GJ, Harmon, LJ, Wegmann D, Joyce, P., Revell, LJ, Alfaro ME. in press. Evolution, also Wegmann D, Leuenberger C, Excoffier L. 2009. Genetics 182:1207-1218. Harmon LJ et al. 2010. Early Bursts of body size and shape evolution are rare in comparative data. Evolution 64: 2385 - 2396

Examples

```
example(mecca)
```

`calibrate.rjmcmc` *initialize proposal width*

Description

estimating a reasonable proposal width to initiate sampling for Markov sampling

Usage

```
calibrate.rjmcmc(phy, dat, nstep = 10000, widths = 2^(-3:3), type=c("bm",
    "rbm", "jump-bm", "jump-rbm"), ...)
```

Arguments

`phy`	a phylogenetic tree of class 'phylo'
`dat`	a named vector of continuous trait values, associated with each species in `phy`
`nstep`	number of proposal steps over which to assess proposal widths
`widths`	if unspecified, a series of proposal widths from 1/8 to 8 will be considered
`type`	a model type available in `rjmcmc.bm`
`...`	arguments to be passed to `make.gbm` and `rjmcmc.bm`

Details

This function may be useful for constraining subsequent runs after an adequate proposal width has been approximated. MCMC samples from this calibration are not stored and do not become available to the user. This function is solely used to give the user a sense of acceptance rates that can be expected for different proposal widths. The narrower the width, the more easily the chain can become stuck. With a wider width, the chain will more quickly explore a broader parameter space, yet acceptance rates may become unacceptably low.

Author(s)

JM Eastman

See Also

`rjmcmc.bm`

Examples

```
n=40
phy=rcoal(n=n)
dat=rTraitCont(phy=phy, model="BM", sigma=sqrt(0.1))
r=paste(sample(letters,9,replace=TRUE),collapse="")

## calibrate proposal width
calibrate.rjmcmc(phy=phy, dat=dat, nstep=500, widths=2^(-3:0), type="rbm")
```

`congruify.phylo` *ultrametricization of trees from a supplied timetree*

Description

automagically generating secondary calibrations

Usage

```
congruify.phylo(reference, target, taxonomy = NULL, tol = 0,
  scale=c(NA, "PATHd8", "treePL"), ncores=NULL)
```

Arguments

`reference`	an ultrametric tree used to time-scale the `target`
`target`	a phylogram that is sought to be ultrametricized based on the `reference` phylogeny
`taxonomy`	a linkage table between tips of the phylogeny and clades represented in the tree; rownames of 'taxonomy' should be tips found in the phylogeny
`tol`	branching time in `reference` above which secondary constraints will be applied to `target`
`scale`	`NA`, `"PATHd8"` or `"treePL"` (if `PATHd8` or `"treePL"` are available in the R `PATH`)
`ncores`	number of cores to be used

Details

This function uses the `reference` to inform secondary calibrations in the `target`. The primary output is a table of 'congruent' nodes between the `reference` and `target` with associated dates extracted from corresponding nodes in the `reference`.

If multiple trees are supplied as the `reference`, a 'congruification' table is generated for each.

If `scale="PATHd8"`, the `target` will be smoothed by `PATHd8` using the "d8 calculation" (see http://www2.math.su.se/PATHd8/PATHd8manual.pdf). This scaling method requires that `PATHd8` is available on the user's `PATH` variable that can be accessed by `Sys.getenv("PATH")`.

If `scale="treePL"`, the `target` will be smoothed by `treePL`. This scaling method requires that `treePL` is available on the user's `PATH` variable that can be accessed by `Sys.getenv("PATH")`.

Author(s)

JM Eastman

References

Eastman JM, LJ Harmon, and DC Tank. 2013. Congruification: support for time scaling large phylogenetic trees. *Methods in Ecology and Evolution*, in press.

Examples

```
## Not run:
sal=get(data(caudata))
res=congruify.phylo(sal$fam, sal$phy, sal$tax, tol=0, scale=NA, ncores=2)
print(res$calibrations)
plot(ladderize(sal$phy,right=FALSE), cex=0.35, type="fan", label.offset=2.5)
plot(ladderize(sal$fam,right=FALSE), cex=0.5, type="fan", label.offset=2.5, no.margin=FALSE)

# if you have PATHd8 installed you can also run
# res=congruify.phylo(sal$fam, sal$phy, sal$tax, tol=0, scale=\"PATHd8\")
# print(res)

## End(Not run)
```

---

dcount *prior densities for truncated discrete random variable*

Description

creating a prior density function for a truncated discrete random variable

Usage

```
dcount(x, FUN, ...)
```

Arguments

x	an integer vector spanning from the minimal to maximal values
FUN	a probability density function (see **Examples**)
...	additional arguments to be passed to FUN

Author(s)

JM Eastman

See Also

make.gbm

Examples

```
range=0:100
u=dcount(range, FUN=dunif, min=min(range), max=max(range))
g=dcount(range, FUN=dgeom, prob=0.05)
p=dcount(range, FUN=dtpois, min=min(range), max=max(range), lambda=0.5)
priors=list(pois=p, geom=g, unif=u)
plot(range(range), c(0,1), xlab="index", ylab="cumsum(prob)", type="n", bty="n")
for(i in 1:length(priors)){
 points(attributes(attributes(priors[[i]])$density)$cumsum, col=i, pch=22, cex=0.35)
}
legend("bottomright", bty="n", legend=names(priors), col=1:length(priors), pch=22)


## LN prior probabilities
print(u(0)) ## dunif
print(g(0)) ## dgeom
print(p(0)) ## dtpois
```

---

drop.extinct *prune specified taxa from a phylogenetic tree*

Description

pruning a set of taxa from a tree

Usage

```
drop.extinct(phy, tol = NULL)
drop.random(phy, n)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
tol	rounding-error tolerance for taxa that do not reach the present day exactly
n	number of random taxa to prune from the tree

Details

The functions prune taxa from a tree either at random or based either on a temporal criterion (whether the leaves reach the present within a given `tol`). By default, `tol = min(phy$edge.length)/100`. The result is a tree that has been pruned based on the given criterion. The function `is.extinct` will return a vector of the tip labels of inferred extinct taxa (or NULL is no extinct taxa exist).

Author(s)

LJ Harmon, and JW Brown

See Also

drop.tip

Examples

```
# Birth-death tree with extinct taxa
p1 <- sim.bdtree(b=0.2, d=0.1, stop="time", seed=1, t=30)
plot(p1, cex=0.25)

# List extinct taxa
p1.extinct <- is.extinct(p1)

# Previous tree with extinct taxa removed
p2 <- drop.extinct(p1)
plot(p2, cex=0.5)
```

---

dtt *disparity-through-time*

Description

calculating and plotting disparity-through-time for a phylogenetic tree and phenotypic data

Usage

```
disparity(phy=NULL, data, index = c("avg.sq", "avg.manhattan", "num.states"))
dtt(phy, data, index=c("avg.sq", "avg.manhattan", "num.states"),
    mdi.range=c(0,1), nsim=0, CI=0.95, plot=TRUE, calculateMDIp=F)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
data	a named vector or matrix of continuous trait values, associated with species in phy
index	disparity index to use (see **Details**)
mdi.range	time range over which to calculate MDI statistic
nsim	number of simulations used to calculate null disparity-through-time plot (see **Details**)
CI	confidence level from which to plot simulated disparities
plot	whether to plot disparities through time
calculateMDIp	calculate p-value for MDI compared to null; only valid if nsim is not zero

Details

The most complete implementation of `dtt` (where `nsim` is greater than 0) carries out the entire disparity-through-time (DTT) procedure described in Harmon et al. 2003, where simulated data are used to construct a null DTT distribution. The function `disparity` simply computes the morphological disparity for a set of species. Note that for `mdi.range`, time is relative to a total tree length of 1. The default `mdi.range` is the entire temporal span of the tree, from 0 (root) to 1 (tips).

For either function, the disparity index can be one of the following:

avg.sq is average squared Euclidean distance among all pairs of points; this is the most common distance metric for disparity in macroevolution

avg.manhattan is average Manhattan distance among all pairs of points

num.states is number of unique character states; this is currently the only option for discrete character data

Value

The function `disparity` returns the disparity of the supplied `data`. If given a tree, `disparity` will return disparities computed for each subtree. The vector of disparities is indexed based on the numeric node-identifier of the subtending subtree (e.g., the root is indexed by N+1, where N is the number of species in the tree; see `read.tree`).

The function `dtt` returns elements from the list below:

dtt is average disparity for clades whose stem crosses each time interval in the tree

times are times for each value in disparity-through-time calculation; these are just the branching times of the phylogeny

sim are disparities at time intervals for each simulated dataset

MDI is the value of the MDI statistic, which is the area between the DTT for the data and the median of the simulations

If results are plotted, the mean DTT from the simulated datasets appears in dashed line and the empirical DTT in solid line.

Author(s)

LJ Harmon and GJ Slater

References

Foote M. 1997. The evolution of morphological diversity. *ARES* 28:129-152.

Harmon LJ, JA Schulte, JB Losos, and A Larson. 2003. Tempo and mode of evolutionary radiation in iguanian lizards. *Science* 301:961-964.

Slater GJ, SA Price, F Santini, and MA Alfaro. 2010. Diversity vs disparity and the evolution of modern cetaceans. *PRSB* 277:3097 -3104.

Examples

```
## Not run:
geo=get(data(geospiza))

## disparity -- not tree-based
disparity(data=geo$dat) # not tree-based

## cladewise disparities
disparity(phy=geo$phy, data=geo$dat)

## disparity through time of culmen length
dttcul<-dtt(phy=geo$phy, data=geo$dat[,"culmenL"], nsim=100, plot=TRUE)

## disparity through time of entire dataset -- without simulated data
dttgeo<-dtt(phy=geo$phy, data=geo$dat, nsim=0, plot=TRUE)

## End(Not run)
```

`fitContinuous` *Model fitting for continuous comparative data*

Description

fitting macroevolutionary models to phylogenetic trees

Usage

```
fitContinuous(phy, dat, SE = 0,
   model = c("BM","OU","EB","rate_trend","lambda","kappa","delta","mean_trend","white"),
    bounds= list(), control = list(method = c("subplex","L-BFGS-B"),
    niter = 100, FAIL = 1e+200, hessian = FALSE, CI = 0.95), ncores=NULL, ...)
```

Arguments

`phy`	a phylogenetic tree of class phylo
`dat`	data vector for a single trait, with names matching tips in `phy`
`SE`	a single value or named vector of standard errors associated with values in `dat`; if any elements in the vector `SE` are NA, SE will be estimated
`model`	model to fit to comparative data (see **Details**)
`bounds`	range to constrain parameter estimates (see **Details**)
`control`	settings used for optimization of the model likelihood
`ncores`	Number of cores. If `NULL` then number of cores is detected
`...`	additional arguments to be passed to the internal likelihood function `bm.lik`

Details

This function fits various likelihood models for continuous character evolution. The function returns parameter estimates and the likelihood for univariate datasets.

The model likelihood is maximized using methods available in `optim` as well as `subplex`. Optimization methods to be used within `optim` can be specified through the `control` object (i.e., `control$method`).

A number of random starting points are used in optimization and are given through the `niter` element within the `control` object (e.g., `control$niter`). The `FAIL` value within the `control` object should be a large value that will be considerably far from -lnL of the maximum model likelihood. In most cases, the default setting for `control$FAIL` will be appropriate. The Hessian may be used to compute confidence intervals (`CI`) for the parameter estimates if the `hessian` element in `control` is TRUE.

Beware: difficulty in finding the optimal solution is determined by an interaction between the nature and complexity of the likelihood space (which is data- and model-dependent) and the numerical optimizer used to explore the space. There is never a guarantee that the optimal solution is found, but using many random starting points (`control$niter`) and many optimization methods (`control$method`) will increase these odds.

Bounds for the relevant parameters of the fitted model may be given through the `bounds` argument. Bounds may be necessary (particularly under the `OU` model) if the likelihood surface is characterized by a long, flat ridge which can be exceedingly difficult for optimization methods. Several bounds can be given at a time (e.g., `bounds=list(SE=c(0,0.1),alpha=c(0,1))` would constrain measurement error as well as the 'constraint' parameter of the Ornstein-Uhlenbeck model). Default bounds under the different models are given below.

Possible models are as follows:

BM is the Brownian motion model (Felsenstein 1973), which assumes the correlation structure among trait values is proportional to the extent of shared ancestry for pairs of species. Default bounds on the rate parameter are `sigsq=c(min=exp(-500),max=exp(100))`. The same bounds are applied to all other models, which also estimate `sigsq`

OU is the Ornstein-Uhlenbeck model (Butler and King 2004), which fits a random walk with a central tendency with an attraction strength proportional to the parameter `alpha`. The `OU` model is called the `hansen` model in **ouch**, although the way the parameters are fit is slightly different here. Default bounds are `alpha = c(min = exp(-500), max = exp(1))`

EB is the Early-burst model (Harmon et al. 2010) and also called the `ACDC` model (accelerating-decelerating; Blomberg et al. 2003). Set by the `a` rate parameter, `EB` fits a model where the rate of evolution increases or decreases exponentially through time, under the model r[t] = r[0] * exp(a * t), where `r[0]` is the initial rate, `a` is the rate change parameter, and `t` is time. The maximum bound is set to `-0.000001`, representing a decelerating rate of evolution. The minimum bound is set to log(10^-5)/depth of the tree.

rate_trend is a diffusion model with linear trend in rates through time (toward larger or smaller rates). Used to be denominated the `"trend"` model, which is still accepted by `fitContinuous` for backward compatibility. Default bounds are `slope = c(min = -100, max = 100)`

lambda is one of the Pagel (1999) models that fits the extent to which the phylogeny predicts covariance among trait values for species. The model effectively transforms the tree: values of `lambda` near 0 cause the phylogeny to become more star-like, and a `lambda` value of 1 recovers the `BM` model. Default bounds are `lambda = c(min = exp(-500), max = 1`

kappa is a punctuational (speciational) model of trait evolution (Pagel 1999), where character divergence is related to the number of speciation events between two species. Note that if there are speciation events that are missing from the given phylogeny (due to extinction or incomplete sampling), interpretation under the kappa model may be difficult. Considered as a tree transformation, the model raises all branch lengths to an estimated power (kappa). Default bounds are `kappa = c(min = exp(-500), max = 1)`

delta is a time-dependent model of trait evolution (Pagel 1999). The `delta` model is similar to `ACDC` insofar as the `delta` model fits the relative contributions of early versus late evolution in the tree to the covariance of species trait values. Where `delta` is greater than 1, recent evolution has been relatively fast; if `delta` is less than 1, recent evolution has been comparatively slow. Intrepreted as a tree transformation, the model raises all node depths to an estimated power (`delta`). Default bounds are `delta = c(min = exp(-500), max = 3)`

mean_trend is a model of trait evolution with a directional drift or *trend* component (i.e., toward smaller or larger values through time). This model is sensible only for non-ultrametric trees, as the likelihood surface is entirely flat with respect to the slope of the trend if the tree is ultrametric. The model used to be denominated the `"drift"` model, which is still accepted by `fitContinuous` for backward compatibility. Default bounds are `drift = c(min = -100, max = 100)`

white is a `white`-noise (non-phylogenetic) model, which assumes data come from a single normal distribution with no covariance structure among species. The variance parameter `sigsq` takes the same bounds defined under the `BM` model

Value

`fitContinuous` returns a list with the following four elements:

`lik` is the function used to compute the model likelihood. The returned function (`lik`) takes arguments that are necessary for the given model. For instance, if estimating a Brownian-motion model with unknown standard error, the arguments (`pars`) to the `lik` function would be `sigsq` and `SE`. By default, the function evaluates the likelihood of the model by assuming the maximum likelihood root state. This behavior can be changed in the call to `lik` with `lik(pars, root=ROOT.GIVEN)` where `pars` includes a value for the root state (z0). See **Examples** for a demonstration. The tree and data are stored internally within the `lik` function, which permits those elements to be efficiently reused when computing the likelihood under different parameter values

`bnd` is a matrix of the used bounds for the relevant parameters estimated in the model. Warnings will be issued if any parameter estimates occur at the supplied (or default) parameter bounds

`res` is a matrix of results from optimization. Rownames of the `res` matrix are the optimization methods (see `optim` and `subplex`). The columns in the `res` matrix are the estimated parameter values, the estimated model likelihood, and an indication of optimization convergence. Values of convergence not equal to zero are not to be trusted

`opt` is a list of the primary results: estimates of the parameters, the maximum-likelihood estimate (`lnL`) of the model, the optimization method used to compute the MLE, the number of model parameters (`k`, including one parameter for the

root state), the AIC (aic), sample-size corrected AIC (aicc). The number of observations for AIC computation is taken to be the number of trait values observed. If the Hessian is used, confidence intervals on the parameter estimates (CI) and the Hessian matrix (hessian) are also returned

Note

To speed the likelihood search, one may set an environment variable to make use of parallel processing, used by mclapply. To set the environment variable, use options(mc.cores=INTEGER), where INTEGER is the number of available cores. Alternatively, the mc.cores variable may be preset upon the initiation of an R session (see Startup for details).

Author(s)

LJ Harmon, W Challenger, and JM Eastman

References

Blomberg SP, T Garland, and AR Ives. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. *Evolution* 57:717-745.

Butler MA and AA King, 2004. Phylogenetic comparative analysis: a modeling approach for adaptive evolution. *American Naturalist* 164:683-695.

Felsenstein J. 1973. Maximum likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics* 25:471-492.

Harmon LJ et al. 2010. Early bursts of body size and shape evolution are rare in comparative data. *Evolution* 64:2385-2396.

Pagel M. 1999. Inferring the historical patterns of biological evolution. *Nature* 401:877-884

Examples

```
## Not run:
geo=get(data(geospiza))

tmp=treedata(geo$phy, geo$dat)
phy=tmp$phy
dat=tmp$data

#---- STORE RESULTS
brownFit <-  fitContinuous(phy, dat[,"wingL"], SE=NA, control=list(niter=50), ncores=2)

#---- PRINT RESULTS
print(names(brownFit))
print(brownFit)

\donttest{
#---- COMPUTE LIKELIHOOD
flik=brownFit$lik
print(argn(flik))

#---- CREATE a FUNCTION to COMPARE MODELS
```

```
fitGeospiza=function(trait=c("wingL","tarsusL","culmenL","beakD","gonysW")){

 trait=match.arg(trait, c("wingL","tarsusL","culmenL","beakD","gonysW"))

 # define set of models to compare
 models=c("BM", "OU", "EB", "white")
 summaries=c("diffusion", "Ornstein-Uhlenbeck", "early burst", "white noise")

 ## ESTIMATING measurement error ##
 aic.se=numeric(length(models))
 lnl.se=numeric(length(models))

 for(m in 1:length(models)){
  cat("\n\n\n\n\t*** ", paste(toupper(summaries[m]),": fitting ", sep=""), models[m],
   " with SE *** \n", sep="")
  tmp=fitContinuous(phy,dat[,trait],SE=NA, model=models[m],
                                    bounds=list(SE=c(0,0.5)), ncores=2)
  print(tmp)
  aic.se[m]=tmp$opt$aicc
  lnl.se[m]=tmp$opt$lnL
 }


 ## ASSUMING no measurement error ##
 aic=numeric(length(models))
 lnl=numeric(length(models))

 for(m in 1:length(models)){
  cat("\n\n\n\n\t*** ", paste(toupper(summaries[m]),": fitting ", sep=""), models[m],
    " *** \n", sep="")
  tmp=fitContinuous(phy,dat[,trait],SE=0,model=models[m], ncores=2)
  print(tmp)
  aic[m]=tmp$opt$aicc
  lnl[m]=tmp$opt$lnL
 }

 ## COMPARE AIC ##
 names(aic.se)<-names(lnl.se)<-names(aic)<-names(lnl)<-models
 delta_aic<-function(x) x-x[which(x==min(x))]

 # no measurement error
 daic=delta_aic(aic)
 cat("\n\n\n\t\t\t\t*** MODEL COMPARISON: ",trait," *** \n",sep="")
 cat("\tdelta-AIC values for models assuming no measurement error
    \t\t\t\t zero indicates the best model\n\n")
 print(daic, digits=2)

  # measurement error
 daic.se=delta_aic(aic.se)
 cat("\n\n\n\n\t\t\t\t\t*** MODEL COMPARISON: ",trait," ***\n",sep="")
 cat("\t\t   delta-AIC values for models estimating SE
    \t\t\t\t zero indicates the best model\n\n")
 print(daic.se, digits=2)
```

```
 cat("\n\n\n")

 res_aicc=rbind(aic, aic.se, daic, daic.se)
 rownames(res_aicc)=c("AICc","AICc_SE","dAICc", "dAICc_SE")

 return(res_aicc)
}

#---- COMPARE MODELS for WING LENGTH
res=fitGeospiza("wingL")
print(res)
}

## End(Not run)
```

`fitContinuousMCMC` *Fit models of continuous trait evolution to comparative data using MCMC*

Description

This function is essentially an MCMC version of the fitContinuous function that fits models using Maximum Likelihood. The main advantage of using MCMC is that informative prior distributions may be placed on node values when such information is available, for example from the fossil record. In such cases, model selection may be improved upon and differ from results using extant taxa only.

Usage

```
fitContinuousMCMC(phy, d, model = c("BM", "Trend", "SSP", "ACDC.exp","ACDC.lin"),
 Ngens = 1e+06, sampleFreq = 1000, printFreq = 1000,
 propwidth = rep(0.1, 5), node.priors = NULL, root.prior = NULL,
 acdc.prior = NULL, sample.node.states = T, outputName = "mcmc.output")
```

Arguments

`phy`	A phylogenetic tree in phylo format
`d`	A named numeric vector of tip values
`model`	The type of model to be fitted. Options are: "BM" - Brownian motion, "Trend" = BM with a directional trend (this option can only be used with non ultrametric trees or when node priors are available for an ultrametric tree), "SSP" - the single stationary peak, or single-optimum OU model, "ACDC.exp" - Accelerating-Decelerating evolution where the change in rate is exponential with respect to time. The DC portion of this model corresponds to the EB model in Harmon et al. (2010), "ACDC.lin" - Accelerating-Decelerating evolution where the change in rate is linear with respect to time (see the supplementary information for Harmon et al. 2010 for details).
`Ngens`	The number of generations that the MCMC should be run for.

`sampleFreq` The frequency with which likelihoods and parameters should be sampled from the chain. For large trees and/or if ancestral states are being sampled, this value should be large (at least equal to the number of internal nodes in the tree).

`printFreq` The frequency with which progress should be printed to the screen.

`propwidth` A numeric vector containing five values corresponding to proposal widths. The order is: 1 - root state proposal width, 2 - rate proposal width, 3 - scaler (alpha, rate change, trend parameter) proposal width, 4 - Theta (OU optimum) proposal width (not yet implemented), and 5- node state proposal width. 5 Values should be provided regardless of the model being fitted. Dummy or random values can be used for non-applicable parameters in such cases.

`node.priors` A data frame of informative node priors. The default is node.priors=NULL, in which case uniformative priors will be assumed for all nodes. Alternatively, the user may provide a dataframe with five columns. The first two columns should specify the tips spanning the clade descended from the node on which an informative prior is to be placed. Columns 3 and 4 give parameters related to the prior while column 5 specifies the type of prior distribution. Current options, along with the associated parameters required in columns 3 and 4, are "normal" (mean and sd), "uniform" (min and max), and "exp" (offset and rate).

`root.prior` A prior for the root value. The default is root.prior = NULL, in which case an uniformative root prior is assumed. Alternatively, the user may provide a vector containing 3 values: 1 & 2 are the parameters for the root prior parameter values, while the 3rd is the type of prior. Current prior types are the same as for node priors.

`acdc.prior` For the ACDC models, it is wise to bound the values of the rate change parameter as unreasonable proposed values can result in rounding issues and negatively infinite log-likelihoods. The default for this option is NULL. If an ACDC model is selected and no prior is given, a bounded, informative uniform prior will be assigned based on the root height of the tree. Alternatively, the user may provide bounds of their own. This may be particularly useful if the user is only interested in a subset of the parameter space area (eg., EB).

`sample.node.states` Logical. If sample.node.states = TRUE, node states (ancestral state values) will be logged to a separate output file. If sample.node.states = FALSE, ancestral states will be treated as nuisance parameters and discarded after computation of likelihoods.

`outputName` stem name for output file(s)

Details

Unlike fitContinuous, which returns output as a list, fitContinuousMCMC outputs directly to text file(s). The model parameter file is designed to be compatatible with the popular Tracer software used in conjunction with BEAST. The output can easily be read back into R for further analysis. A particular advantage of writing directly to text file is that crashes etc do not result in the complete loss of long runs, and memory is saved.

Value

If sample.node.states = FALSE, one text file is written to the current working directory.

`outputName_model_params.txt`
: a text file containing sampled prior, posterior, likelihood, and model parameter values from the MCMC run

If sample.node.states = TRUE, a second text file is written

`outputName_nodestates.txt`
: contains the posterior sample of ancestral states for internal nodes

Author(s)

Graham Slater

References

Slater GJ, LJ Harmon, and ME Alfaro. 2012. Integrating fossils with molecular phylogenies improves inference of trait evolution. *Evolution* 66:3931-3944.

Examples

```
## Not run:
data(caniformia)
phy <- caniformia$phy
d <- caniformia$dat
node.priors <- caniformia$node.priors
root.prior <- caniformia$root.prior

## as an example, we will run a very short (too short!) analysis,
##fitting BM and Trend to the caniform data

fitContinuousMCMC(phy, d, model = "BM", Ngens = 1000, sampleFreq =100,
printFreq = 100, node.priors = node.priors, root.prior = root.prior,
outputName ="BM_caniforms")

fitContinuousMCMC(phy, d, model = "Trend", Ngens = 1000, sampleFreq=100,
printFreq = 100, node.priors = node.priors, root.prior = root.prior,
outputName ="Trend_caniforms")

bm.res <- read.table("BM_caniforms_model_params.txt", header= TRUE)
head(bm.res)

trend.res <- read.table("Trend_caniforms_model_params.txt", header= TRUE)
head(trend.res)

### produce trace plots of logLk scores

plot(bm.res$generation, bm.res$logLk, type = "l",
ylim = c(min(bm.res$logLk), max = max(trend.res$logLk)))

lines(trend.res$generation, trend.res$logLk, col = "red")
legend("bottomleft", c("bm", "trend"), lwd = 3, col = c("black", "red"))

## End(Not run)
```

fitDiscrete *Model fitting for discrete comparative data*

Description

fitting macroevolutionary models to phylogenetic trees

Usage

```
fitDiscrete(phy, dat,
    model = c("ER","SYM","ARD","meristic"),
    transform = c("none", "EB", "lambda", "kappa", "delta", "white"),
    bounds = list(), control = list(method = c("subplex", "L-BFGS-B"),
    niter = 100, FAIL = 1e+200, hessian = FALSE, CI = 0.95), ncores=NULL,
    ...)
## S3 method for class 'gfit'
as.Qmatrix(x, ...)
```

Arguments

phy	a phylogenetic tree of class phylo
dat	data vector for a single trait, with names matching tips in phy
model	an Mkn model to fit to comparative data (see **Details**)
transform	an evolutionary model used to transform the tree (see **Details**)
bounds	range to constrain parameter estimates (see **Details**)
control	settings used for optimization of the model likelihood
ncores	Number of cores. If NULL then number of cores is detected
x	Object of class "gfit" for S3 method as.Qmatrix
...	if model="meristic", ... can dictate whether the matrix is asymmetric (symmetric=FALSE)

Details

This function fits various likelihood models for discrete character evolution. The function returns parameter estimates and the likelihood for univariate datasets. All of the models are continuous-time Markov models of trait evolution (see Yang 2006 for a good general discussion of this type of model).

The model likelihood is maximized using methods available in optim as well as subplex. Optimization methods to be used within optim can be specified through the control object.

A number of random starting points are used in optimization and are given through the niter element within the control object (e.g., control$niter). Finding the maximum likelihood fit is sometimes tricky, especially as the number of parameters in the model increases. Even in the example below, a slightly suboptimal fit is occasionally returned with the default settings fitting the general (ARD) model. There is no rule of thumb for the number of iterations that will be appropriate for a given dataset and model, but one use the variance in fitted likelihoods across iterations as an

indication of the difficulty of the likelihood space (see details of the `res` object in **Value**). Twenty optimization iterations per parameter seems to be a decent *starting* point for fitting these models.

The `FAIL` value within the `control` object should be a large value that will be considerably far from -lnL of the maximum model likelihood. In most cases, the default setting for `control$FAIL` will be appropriate. The Hessian may be used to compute confidence intervals (`CI`) for the parameter estimates if the `hessian` element in `control` is TRUE.

The function can handle traits with any number of character states, under a range of models. The character model is specified by the `model` argument:

ER is an `equal-rates` model of where a single parameter governs all transition rates

SYM is a `symmetric` model where forward and reverse transitions share the same parameter

ARD is an `all-rates-different` model where each rate is a unique parameter

meristic is a model wherein transitions occur in a stepwise fashion (e.g., 1 to 2 to 3 to 2) without skipping intermediate steps; this requires a sensible coding of the character states as consecutive integers are assumed to be neighboring states

matrix is a user supplied model (given as a dummy matrix representing transition classes between states); elements that are zero signify rates that are also zero (see **Examples**)

The `transform` argument allows one to test models where rates vary across the tree. Bounds for the relevant parameters of the tree `transform` may be given through the `bounds` argument. Several bounds can be given at a time. Default bounds under the different models are given below. Options for `transform` are as follows:

none is a model of rate constancy through time

EB is the Early-burst model (Harmon et al. 2010) and also called the `ACDC` model (accelerating-decelerating; Blomberg et al. 2003). Set by the `a` rate parameter, `EB` fits a model where the rate of evolution increases or decreases exponentially through time, under the model r[t] = r[0] * exp(a * t), where `r[0]` is the initial rate, `a` is the rate change parameter, and `t` is time. Default bounds are `a = c(min = -10, max = 10)`

lambda is one of the Pagel (1999) models that fits the extent to which the phylogeny predicts covariance among trait values for species. The model effectively transforms the tree: values of `lambda` near 0 cause the phylogeny to become more star-like, and a `lambda` value of 1 recovers the `none` model. Default bounds are `lambda = c(min = 0, max = 1)`

kappa is a punctuational (speciational) model of trait evolution (Pagel 1999), where character divergence is related to the number of speciation events between two species. Note that if there are speciation events in the given phylogeny (due to extinction or incomplete sampling), interpretation under the `kappa` model may be difficult. Considered as a tree transformation, the model raises all branch lengths to an estimated power (`kappa`). Default bounds are `kappa = c(min = 0, max = 1)`

delta is a time-dependent model of trait evolution (Pagel 1999). The `delta` model is similar to `ACDC` insofar as the `delta` model fits the relative contributions of early versus late evolution in the tree to the covariance of species trait values. Where `delta` is greater than 1, recent evolution has been relatively fast; if `delta` is less than 1, recent evolution has been comparatively slow. Intrepreted as a tree transformation, the model raises all node depths to an estimated power (`delta`). Default bounds are `delta = c(min = 0, max = 3)`

white is a `white`-noise (non-phylogenetic) model, which converts the tree into a star phylogeny

Value

`fitDiscrete` returns a list with the following four elements:

`lik` is the function used to compute the model likelihood. The returned function (`lik`) takes arguments that are necessary for the given model. For instance, if estimating an untransformed `ER` model, there would be a single argument (the transition rate) necessary for the `lik` function. The tree and data are stored internally within the `lik` function, which permits those elements to be efficiently reused when computing the likelihood under different parameter values. By default, the function evaluates the likelihood of the model by weighting root states in accordance with their conditional probability given the data (this is the `"obs"` option; see FitzJohn et al. 2009). This default behavior can be changed in the call to `lik` with `lik(pars, root="flat")`, for instance, which would weight each state equally at the root. The other useful option is `"given"`, where the user must also supply a vector (`root.p`) of probabilities for each possible state. To make likelihoods roughly comparable between **geiger** and **ape**, one should use the option `lik(pars, root="given", root.p=rep(1,k))`, where `k` is the number of character states. See **Examples** for a demonstration

`bnd` is a matrix of the used bounds for the relevant parameters estimated in the model. Warnings will be issued if any parameter estimates occur at the supplied (or default) parameter bounds

`res` is a matrix of results from optimization. Rownames of the `res` matrix are the optimization methods (see `optim` and `subplex`). The columns in the `res` matrix are the estimated parameter values, the estimated model likelihood, and an indication of optimization convergence. Values of convergence not equal to zero are not to be trusted

`opt` is a list of the primary results: estimates of the parameters, the maximum-likelihood estimate (`lnL`) of the model, the optimization method used to compute the MLE, the number of model parameters (`k`, including one parameter for the root state), the AIC (`aic`), sample-size corrected AIC (`aicc`). The number of observations for AIC computation is taken to be the number of trait values observed. If the Hessian is used, confidence intervals on the parameter estimates (`CI`) and the Hessian matrix (`hessian`) are also returned

Note

To speed the likelihood search, one may set an environment variable to make use of parallel processing, used by `mclapply`. To set the environment variable, use `options(mc.cores=INTEGER)`, where `INTEGER` is the number of available cores. Alternatively, the `mc.cores` variable may be preset upon the initiation of an R session (see `Startup` for details).

Author(s)

LJ Harmon, RE Glor, RG FitzJohn, and JM Eastman

References

Yang Z. 2006. *Computational Molecular Evolution*. Oxford University Press: Oxford. FitzJohn RG, WP Maddison, and SP Otto. 2009. Estimating trait-dependent speciation and extinction rates from incompletely resolved molecular phylogenies. *Systematic Biology* 58:595-611.

Examples

```
## Not run:
## match data and tree
tmp=get(data(geospiza))
td=treedata(tmp$phy, tmp$dat)
geo=list(phy=td$phy, dat=td$data)
gb=round(geo$dat[,5]) ## create discrete data
names(gb)=rownames(geo$dat)

tmp=fitDiscrete(geo$phy, gb, model="ER", control=list(niter=5), ncores=2) #-7.119792
## using the returned likelihood function
lik=tmp$lik
lik(0.3336772, root="obs") #-7.119792
lik(0.3336772, root="flat") #-8.125354
lik(0.3336772, root="given", root.p=rep(1/3,3)) #-8.125354
lik(0.3336772, root="given", root.p=c(0, 1, 0)) #-7.074039
lik(c(0.3640363), root="given", root.p=rep(1,3)) #-7.020569 & comparable to ape:::ace solution

## End(Not run)

# general model (ARD)
## match data and tree
tmp=get(data(geospiza))
td=treedata(tmp$phy, tmp$dat)
geo=list(phy=td$phy, dat=td$data)
gb=round(geo$dat[,5]) ## create discrete data
names(gb)=rownames(geo$dat)
fitDiscrete(geo$phy, gb, model="ARD", ncores=1) #-6.064573

# user-specified rate classes
mm=rbind(c(NA, 0, 0), c(1, NA, 2), c(0, 2, NA))
fitDiscrete(geo$phy, gb, model=mm, ncores=1) #-7.037944

# symmetric-rates model
fitDiscrete(geo$phy, gb, model="SYM", ncores=1)#-6.822943
```

---

gbresolve *NCBI taxonomy*

Description

working with NCBI taxonomy

Usage

```
gbresolve(x, rank="phylum", within="", ncores=1, ...)
gbcontain(x, rank="species", within="", ncores=1,...)
```

Arguments

x	a phylogenetic tree of class 'phylo' (gbresolve.phylo), or a string vector (gbresolve), or a single string (gbcontain)
rank	a Linnaean rank to restrict taxonomic resolution
within	a character string representing a group within which to resolve query
ncores	number of cores to use.
...	additional arguments to be passed to the taxdump constructor (ncbit)

Details

The functions access the NCBI taxonomy resource (https://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/; see also ncbit). gbresolve resolves the taxonomic hierarchy for queried taxa up to the given rank (or between the given ranks if two are given), and gbcontain resolves all taxa found within a given queried group and occurring at a specified rank. The rank must be found within the object Linnaean (see **Examples**). The argument within can restrict the group within which to conduct the search (see **Examples**).

The local copy of the taxonomy resource (accessible in with data(ncbi) from **ncbit**) can be updated with a call to ncbit(update=TRUE).

Setting the ncores argument to NULL will use all available cores.

Author(s)

JM Eastman

Examples

```
## call up NCBI taxonomy
ncbi=ncbit::ncbit(update=FALSE)

## possible ranks
print(Linnaean)

## resolve taxa
gbresolve(c("Ambystoma_laterale", "Dicamptodon_copei"))
gbresolve("Andrias japonicus")


## resolve taxa found in tree
sal=get(data(caudata))
x=gbresolve(sal$phy, rank=c("genus", "order"))
plot(x$phy, show.node=TRUE, type="f", show.tip=FALSE, cex=0.3)

## find all genera within salamanders
```

```
gbcontain("caudata", rank="genus")
```

`geiger-data` *example datasets*

Description

providing access to comparative datasets

Usage

```
data(amphibia)
data(caniformia)
data(carnivores)
data(caudata)
data(chelonia)
data(geospiza)
data(primates)
data(whales)
```

Details

The objects `caudata`, `chelonia`, `carnivores`, `geospiza`, and `primates` each have at least two items, a `phy` object and a `dat` object. The `phy` object is a phylogenetic tree of class 'phylo' (see `read.tree`). The `dat` object (e.g., `caudata$dat`) is a named vector of (natural log-transformed) body sizes for each group. The salamander data (object `caudata`) also includes the systematics for all recognized taxa (object `caudata$tax`) as well as a time-calibrated family-level phylogeny (`caudata$fam`) from Zhang and Wake (2009). The object `caudata$phy` is an unpublished glomogram of mostly family level phylogenies from the literature. The backbone for that tree is from Zhang and Wake (2009).

The `amphibia` object is a set of three trees, the last of which is a time-scaled estimate of the Pyron and Wiens (2011) tree (see `congruify.phylo`).

The `whales` object is a dataset including a tree and a taxon richness matrix (see `medusa`).

References

Data are from the following sources:

`AMPHIBIANS (amphibia)`

- Roelants K, DJ Gower, M Wilkinson, SP Loader, SD Biju, K Guillaume, L Moriau, and F Bossuyt. 2007. Global patterns of diversification in the history of modern amphibians. *PNAS* 104:887-892.
- Pyron RA and JJ Wiens. 2011. A large-scale phylogeny of Amphibia including over 2800 species, and a revised classification of extant frogs, salamanders, and caecilians. *MPE* 61:543-583.

`SALAMANDERS (caudata)`

- Adams DC, CM Berns, KH Kozak, and JJ Wiens. 2009. Are rates of species diversification correlated with rates of morphological evolution? *PRSB* 276:2729-2738.
- Bonett RM, PT Chippindale, PE Moler, RW van Devender, and DB Wake. 2009. Evolution of gigantism in amphiumid salamanders. *PLoSONE* 4(5):e5615.
- Kozak KH, RW Mendyk, and JJ Wiens. 2009. Can Parallel Diversification Occur in Sympatry? Repeated Patterns of Body-Size Evolution in Coexisting Clades of North American Salamanders. *Evolution* 63:1769-1784.
- Weisrock DW, TJ Papenfuss, JR Macey, SN Litvinchuk, R Polymeni, IH Ugurtas, E Zhao, H Jowkar, and A Larson. 2006. A molecular assessment of phylogenetic relationships and lineage accumulation rates within the family Salamandridae (Amphibia, Caudata). *MPE* 41:368-383.
- Wiens JJ and JT Hoverman. 2008. Digit reduction, body size, and paedomorphosis in salamanders. *Evolution and Development* 10:449-463.
- Zhang P, Y-Q Chen, H Zhou, X-L Wang, TJ Papenfuss, DB Wake and L-H Qu. 2006. Phylogeny, evolution, and biogeography of Asiatic salamanders (Hynobiidae). *PNAS* 103:7360-7365.
- Zhang P and DB Wake. 2009. Higher-level salamander relationships and divergence dates inferred from complete mitochondrial genomes. *MPE* 53:492-508.

`PRIMATES (primates)`

- Redding DW, C DeWolff, and AO Mooers. 2010. Evolutionary distinctiveness, threat status and ecological oddity in primates. *Conservation Biology* 24:1052-1058.
- Vos RA and AO Mooers. 2006. A new dated supertree of the Primates. Chapter 5. In: VOS RA (Ed.) *Inferring large phylogenies: the big tree problem*. [Ph.D. thesis]. Burnaby BC, Canada: Simon Fraser University.

`CARNIVORES (carnivores)`

- Eizirik E, WJ Murphy, K-P Koepfli, WE Johnson, JW Dragoo, RK Wayne, and SJ O'Brien. 2010. Pattern and timing of diversification of the mammalian order Carnivora inferred from multiple nuclear gene sequences. *Molecular Phylogenetic and Evolution* 56:49-63.
- Wozencraft WC. 2005. Order Carnivora in Wilson DE and DM Reeder (Eds.) *Mammal Species of the World*. Johns Hopkins University Press.
- Jones KE, J Bielby, M Cardillo, et al. 2009. *Ecological Archives* E090-184.

`CANIFORMS (caniformia)`

- Slater GJ, LJ Harmon, and ME Alfaro. 2012. Integrating fossils with molecular phylogenies improves inference of trait evolution. *Evolution* 66:3931-3944.

`TURTLES (chelonia)`

- Jaffe AL, GJ Slater, and ME Alfaro. 2011. Ecological habitat and body size evolution in turtles. *Biology Letters* 7:558-561.

DARWINS FINCHES (geospiza)

- Schluter D

WHALES (whales)

- data compiled by GJ Slater
- Paleobiology Database. 2011. https://paleobiodb.org/.

---

geiger-defunct *deprecated functions in GEIGER*

Description

This function has been deprecated in **geiger** and **auteur** or may be available from another package. Below shows the original function and the suggested function to use in its place.

Details

- area.between.curves: use geiger:::.area.between.curves
- BDsim: use sim.bd
- birthdeath.tree: use sim.bdtree
- calibrate.proposalwidth: use calibrate.rjmcmc
- deltaTree: use rescale.phylo
- disp.calc: use disparity
- dtt.full: use dtt
- exponentialchangeTree: use rescale.phylo
- get.simulation.matrix: use geiger:::.get.simulation.matrix
- getAncStates: use fastAnc
- ic.sigma: use ratematrix
- intercalate.samples: use geiger:::.intercalate.rjmcmc
- kappaTree: use rescale.phylo
- lambdaTree: use rescale.phylo
- linearchangeTree: use rescale.phylo
- name.check: use geiger:::.treedata
- node.leaves: use tips
- node.sons: use geiger:::.get.desc.of.node
- ouTree: use rescale.phylo
- phy.anova: use aov.phylo
- phy.manova: use aov.phylo
- pool.rjmcmcsamples: use load

- prune.extinct.taxa: use drop.extinct
- prune.random.taxa: use drop.random
- rate.estimate: use bd.ms or bd.km
- rescaleTree: use rescale.phylo
- runMedusa: use medusa
- shifts.plot: use plot
- speciationalTree: use rescale.phylo
- tip.disparity: use disparity
- tracer: use of plot.mcmc and other functions in **coda** recommended
- transform.phylo: use rescale.phylo
- tworateTree: use rescale.phylo
- vmat: use geiger:::.vmat

---

geiger-example *internal geiger functions*

Description

geiger-example functions are used solely for demonstrative purposes

---

load.rjmcmc *posterior samples from single or multiple MCMC runs*

Description

generating a pooled and thinned sample of posterior estimates from rjmcmc

Usage

```
load.rjmcmc(x, phy = NULL, burnin = NULL, thin = NULL, ...)
```

Arguments

x	a vector of directory names (character strings) in which samples generated by **geiger** are found
phy	a phylogenetic tree of class 'phylo' for which to compile results; if NULL, the tree stored within each generated rda file is used
burnin	the proportion of each chain to be removed as burnin
thin	an integer specifying the thinning interval for chains
...	arguments to be passed to other functions (has no effect in the present context)

Details

This function provides a means to compile results from at least a single MCMC run, and is especially useful in pooling of multiple independent runs. In cases where MCMC chains were sampled using a set of (different) trees, the argument phy can be used to summarize compiled results against a single summary tree. Branchwise samples are stored with unique edge identifiers (character strings) that are more reliable than the use of numeric node-identifiers (see digest for the function used to generate these unique edge labels).

If it is desired to run analyses for the same trait data across a set of trees (see **Examples**), it is strongly recommended that the set of trees be sent to rjmcmc.bm as a multiPhylo object (see read.tree).

Value

an object of class rjmcmc or rjmcmcmc

Author(s)

JM Eastman

See Also

rjmcmc.bm

Examples

```
sal=get(data(caudata))
a<-sim<-sal$phy
bl=c(386,387,388,183,184,185,186)
mod=match(bl, sim$edge[,2])
sim$edge.length[mod]=sim$edge.length[mod]*64
dat=rTraitCont(sim)
while(1){
 b=a
 b$tip.label[183:186]=sample(b$tip.label[183:186])
 if(!all(a$tip.label==b$tip.label)) break()
}

trees=list(a=a,b=b, c=ladderize(a, right=TRUE), d=ladderize(a, right=FALSE))
class(trees)="multiPhylo"
rjmcmc.bm(trees, dat, ngen=1e3, type="rbm")

res=load.rjmcmc(paste("relaxedBM", names(trees), sep="."), phy=trees$d, burnin=0.25)
plot(res, par="shifts", show.tip=FALSE, edge.width=2.5)
```

make.gbm *tailor reversible-jump Markov chain Monte Carlo sampling*

Description

controlling reversible-jump Markov chain Monte Carlo sampling

Usage

```
make.gbm(phy, dat, SE=NA, type = c("bm", "rbm", "jump-bm", "jump-rbm"), ...)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
dat	a named vector of continuous trait values, associated with each species in phy
SE	a named vector of standard errors for each trait value; applied to all trait values if given a single value
type	the class of model to use (see rjmcmc.bm)
...	arguments passed internally to control other settings (see **Details**)

Details

The argument ... controls the substitution of default settings for Markov-chain Monte Carlo sampling. Below are the settings that are controllable by the user. These parameters and their default settings can also be found with an *empty* call to the function (e.g., make.gbm()).

measurement error (SE): one of the arguments necessary for running rjmcmc.bm is SE, which is a statement about the error associated with the values given in dat. Measurement error (whose argument is SE) can be a named vector of numeric values (including NA) or a single value (including NA). If given as a vector, SE must have names that correspond to the those found for dat. If given a single value for SE, the sampler will apply that value of measurement error to all tips in the tree. If NA appears for the measurement error for *any* species, SE becomes an additional parameter of the model: this density is consequently sampled by rjmcmc.bm. The default for rjmcmc.bm is to estimate a single SE (which is applied to all species).

control settings: default settings for each control parameter are given below. Note that for the discrete random variables (for which dlnSHIFT and dlnJUMP) apply, certain criteria must be met if the user prefers to supply a different prior density. The function dcount is useful for building a custom prior density function for discrete variables.

Value

The items that can be tailored in the resulting control object are as follows:

\bold{method}	default direct; this defines the implementation for computing model likelihood (currently with only the default available)

\bold{constrainSHIFT}
: default FALSE; if integer given, this constrains the number of local clocks in the diffusive process

\bold{constrainJUMP}
: default FALSE; if integer given, this constrains the number of pulses in the jump process

\bold{dlnSHIFT}
: default dcount(0:(nn - 1), FUN = dpois, lambda = log(2)), where nn is the number of branches in the tree; this controls the prior density on the number of shifts between local clocks in the tree (and applies only if the type argument to rjmcmc.bm is "rbm" or "jump-rbm") and if constrainSHIFT is FALSE)

\bold{dlnJUMP}
: default dcount(0:nn, FUN = dlunif, min = 0, max = nn, dzero = 0.5), where nn is the number of branches in the tree; this controls the prior density on the number of evolutionary pulses across the tree (and applies only if the type argument to rjmcmc.bm is "jump-bm" or "jump-rbm" and if constrainJUMP is FALSE)

\bold{dlnRATE}
: default function (x) dexp(x, rate = 1/(10^3), log = TRUE); this defines the prior density on rate scalars

\bold{dlnSE}
: default function (x) dexp(x, rate = 1/(10^3), log = TRUE); this defines the prior density on measurement error

\bold{dlnPULS}
: default function (x) dexp(x, rate = 1/(10^3), log = TRUE); this defines the prior density on jump variance

\bold{dlnROOT}
: default function (x) dunif(x, min=-10^3, max=10^3, log=TRUE); this defines the prior density on root state

\bold{rate.lim}
: default list(min=0, max=Inf); this defines the numerical limits on the rate scalars

\bold{se.lim}
: default list(min=0, max=Inf); this defines the numerical limits on the measurement error

\bold{root.lim}
: default list(min=-Inf, max=Inf); this defines the numerical limits on the root state

\bold{jump.lim}
: default 1; determines the number of jumps permissible along each branch (1 is currently supported)

\bold{excludeSHIFT}
: default c(); if this argument is not empty, these are branches (specified by numeric node-identifiers) which cannot be chosen for a novel local clock

\bold{excludeJUMP}
: default c(); if this argument if not empty, these are branches (specified by numeric node-identifiers) which cannot be chosen for an evolutionary pulse

\bold{bm.jump}
: default 0.5; this defines the ratio between proposals for the diffusion and jump processes (0.5 is perfectly balanced)

\bold{mergesplit.shift}
: default 0.5; this defines the ratio between proposals that scale model complexity and those that do not

\bold{tune.scale}
: default 0.65; this defines the ratio between proposals that modify the model branchwise versus treewide

\bold{slide.mult}
: default 0.25; this defines the ratio between sliding window and multiplier proposals

\bold{prob.dimension}
: default 0.65; this defines the proportion of proposals used to modify model dimensionality

\bold{prob.effect}
: default 0.3; this defines the proportion of proposals that do not alter dimensionality

\bold{prob.SE}
: default 0.03; this defines the proportion of proposals that alter the (unknown) measurement error

\bold{prob.root}
: default 0.02; this defines the proportion of proposals that alter the root state

\bold{prop.width}
: default 1; this defines the proposal width used for multiplier and sliding-window proposals

\bold{simple.start}
: default TRUE; this determines whether to start the MCMC sampler with minimal dimensionality

\bold{filebase}
: default "result"; this defines a 'base' filename for the output

Author(s)

JM Eastman

See Also

rjmcmc.bm

---

mecca *running a MECCA analysis*

Description

Runs MECCA's hybrid ABC-MCMC algorithm to jointly estimate diversification rates and trait evolution from incompletely sampled comparative data. Many of the arguments taken by this function are the same as those in calibrateMecca().

Usage

```
mecca(phy, richness, cladeMean, cladeVariance, model = c("BM", "Trend", "twoRate"),
  prior.list = list(priorSigma = c(-4.961845, 4.247066), priorMean = c(-10, 10)),
   start = start, Ngens = 10000, printFreq = 100, sigmaPriorType = "uniform",
  rootPriorType = "uniform", SigmaBounds = c(-4.961845, 4.247066), hotclade = NULL,
  divPropWidth = 0.1, scale = 1, divSampleFreq = 0, BoxCox = TRUE, outputName = "mecca")
```

Arguments

phy	time-calibrated phylogenetic tree of class 'phylo'
richness	named vector of species richnesses corresponding to tips in the tree (if a tip does not represent a higher level taxon, its richness should be 1)
cladeMean	named vector of trait means (all tips in the tree must be represented)
cladeVariance	named vector of trait variances (all tips in the tree must be represented; if only one taxon is present, use 0 for the variance)
model	model of trait evolution to be used – options currently implemented are "BM" = Brownian Motion, "Trend" = Brownian moton with a trend, and "twoRate" = two Brownian rate model (see hotclade below).
prior.list	a list containing prior distribution parameters (default values used if this argument is empty)
start	ouput of startingpt.mecca
Ngens	number of generations to run MECCA
printFreq	frequency of printing the acceptance rate
sigmaPriorType	type of prior distribution on the Brownian rate parameter (currently either "uniform" or "normal")
rootPriorType	type of prior distribution on the root state rate parameter (currently "uniform" is available)
SigmaBounds	bounds for sigma (default values correspond to a wide range taken from Harmon et al. 2010)
hotclade	if a two-rate model is to be fit, this specifies which clade takes the second rate – two names should be specified in a vector; either the two tip names that span the clade of interest, or the name of a terminal/internal edge and NULL if only one branch takes the second rate
divPropWidth	proposal width for the diversification MCMC (default value of 0.1 seems to work well)
scale	a numeric value by which the proposal width for trait evolution parameters will be multiplied (a value of 2 seems to work well, but this should be adjusted for each individual dataset)
divSampleFreq	whether new trees are simulated at every step – the default (0) is yes; if a non-zero value is given, this will determine the frequency (every n steps) with which new tip trees are simulated
BoxCox	whether summaries are BOX-COX standardized – default (1) is yes and is recommended; this should always be consistent with the calibration step
outputName	name stem for output file names

Details

The output files produced are formatted to be used with the C++ Program ABCtoolbox (Wegmann et al. 2010), which produces adjusted posterior distributions and can perform model selection without likelihoods.

Value

MECCA does not store any output in memory. Instead, five output files are generated to the current working directory. This files are fully compatible with ABCtoolbox (Wegmann et al. 2011). The first file (outputname_bdSimFile.tx) will output the posterior sample for diversification parameters. The second file (outputname_bmSimFile.txt) ouputs the sampled trait evolution parameters and their associated raw summary statistics while outputname_ObsFile.txt gives the observed summaries.For ABC toolbox though, it will often be more efficient to use pls-transformed versions of the observed and simulated summary statistics. These are available in outputname_distObs.txt, and outputname_distSimFile.txt.

Note

The numbers printed to the screen during the run give the current generation, acceptance rate for diversification parameters, acceptance rate for trait evolutionary rate parameters and acceptance rate for root state parameters, respectively

Author(s)

Graham Slater, Luke Harmon, Daniel Wegmann

References

Slater GJ, LJ Harmon, D Wegmann, P Joyce, LJ Revell, and ME Alfaro. 2012. Fitting models of continuous trait evolution to incompletely sampled comparative data using approximate Bayesian computation. *Evolution* 66:752-762.

Examples

```
## Not run:
data(carnivores)
phy <- carnivores$phy
data <- carnivores$dat
richness <- data[,1]
names(richness) <- rownames(data)

priors <- list(priorSigma = c(-4.5, 4.5), priorMean = c(-5, 2))

## CALIBRATION (far too short for a real analysis)
Cal <- calibrate.mecca(phy, richness, model = "BM", prior.list = priors, Ncalibrations = 1000)

params <- Cal$trait[, c(1,2)] ## extract the calibration BM parameters
stats <- Cal$trait[, -c(1,2)] ## extract the calibration summary stats

## now we run pls, determining combinations of summaries that explain variation in our parameters
## For BM, 2 components is sufficient. For more complex models, more componenets will be required.
require(pls)
myPlsr<-pls::plsr(as.matrix(params) ~ as.matrix(stats), scale=F, ncomp = 2)

plot(RMSEP(myPlsr)) ## Look at Root Mean Square error plots

summary(myPlsr) ## take a look at
```

```
plsdat <- myPlsr$loadings

## extract means and variances from the carnivore data ##

cladeMean<-data[,2]
names(cladeMean)<-rownames(data)
cladeVariance<-data[,3]
names(cladeVariance)<-rownames(data)

## STARTING POINT
## And now we can compute starting values for the ABC-MCMC
start <- startingpt.mecca(Cal, phy, cladeMean, cladeVariance,
tolerance = 0.05, plsdat, BoxCox = TRUE)


## MECCA (far too short for a real analysis)
mecca(phy, richness, cladeMean, cladeVariance, model = "BM", prior.list = priors, start = start,
   Ngens = 1000, printFreq = 100, sigmaPriorType = "uniform", rootPriorType = "uniform",
  SigmaBounds = c(-4.5, 4.5), divPropWidth = 0.1, scale = 2, divSampleFreq = 0, BoxCox = TRUE,
   outputName ="MeccaBM.txt")

## PASTE UNCOMMENTED FOLLOWING LINE TO DROP FILES CREATED BY MECCA
 # unlink(dir(pattern=paste(r)),recursive=TRUE)


## End(Not run)
```

---

medusa *MEDUSA: modeling evolutionary diversification using stepwise AIC*

Description

Fits piecewise birth-death models to ultrametric phylogenetic tree(s) according to phylogenetic (edge-length) and taxonomic (richness) likelihoods. Optimal model size is determined via a stepwise AIC approach.

Usage

```
medusa(phy, richness = NULL, criterion = c("aicc", "aic"),
    partitions=NA, threshold=NA, model = c("mixed", "bd", "yule"),
    cut = c("both","stem","node"), stepBack = TRUE,
    init = c(r=0.05, epsilon=0.5), ncores = NULL, verbose = FALSE, ...)
```

Arguments

phy	an ultrametric phylogenetic tree of class 'phylo'
richness	an optional matrix of taxonomic richnesses (see **Details**)
criterion	an information criterion to use as stopping criterion

partitions	the maximum number of models to be considered
threshold	the improvement in AIC score that should be considered significant (see **Details**)
model	the flavor of piecewise models to consider (see **Details**)
cut	determines where shifts are placed on the tree (see **Details**)
stepBack	determines whether parameter/model removal is considered. default = TRUE. (see **Details**)
init	initial conditions for net diversification and relative extinction
ncores	the number of processor cores to using in parallel processing. default = all
verbose	print out extra information. default = FALSE
...	additional arguments to be passed to treedata

Details

The MEDUSA model fits increasingly complex diversification models to a dataset including richness information for sampled tips in phy. The tree must have branch lengths proportional to time. The richness object is optional, but must be given if the tree is not completely sampled. MEDUSA assumes that the entire extant diversity in the group is sampled either in phy or given by information contained within the richness object. The richness object associates species richness with lineages sampled in the tree. For instance, if a genus containing a total of 10 species is exemplied in the tree by a single tip, the total diversity of the clade must be recorded in the richness object (see **Examples**). All taxa missing from the tree have to be assigned to one of the tips in the richness matrix. If the richness object is NULL, the tree is assumed to be completely sampled.

The algorithm first fits a single diversification model to the entire dataset. A series of single breakpoints in the diversification process is then added, so that different parts of the tree evolve with different parameter values (per-lineage net diversification–r and relative extinction rates–epsilon). Initial values for these diversification parameters are given through the init argument and may need to be tailored for particular datasets. The algorithm compares all single-breakpoint models to the initial model, and retains the best breakpoint. Then all possible two-breakpoint models are compared with the best single-breakpoint model, and so on. Breakpoints may be considered at a "node", a "stem" branch, or both (as dictated by the cut argument). Birth-death or pure-birth (Yule) processes (or a combination of these processes) may be considered by the MEDUSA algorithm. The model flavor is determined through the model argument.

Two stopping criteria are available for the MEDUSA algorithm. The user can either limit the number of piecewise models explored by MEDUSA or this number may be determined based on model fits. A maximum number of model partitions to be explored may be given a priori (if given a nonzero number through the partitions argument) or an information criterion is used to choose sufficient model complexity. If the latter stopping criterion is used, one needs to specify whether to use Akaike information criterion ("aic") or sample-size corrected AIC ("aicc"); the latter is recommended, and is the default setting. An appropriate threshold in AICc differences between different MEDUSA models has been shown to be dependent on tree size. The threshold used for model selection is computed internally (and is based on extensive simulation study); this value is reported to the user. The user may choose to specify an alternative AIC-threshold with the ("threshold") argument, making the algorithm more (or less) strict in scrutinizing model improvement.

The user will almost certainly want to summarize the object returned from MEDUSA with the function TBA.

Value

A list object is returned including fits for all model complexities as well as summary information:

`control`	is a list object specifying the stopping criterion, the information criterion and threshold used (if appropriate), or the number of partitions explored
`cache`	is a list object primarily used internally (including the tree and richness information)
`models`	is a list object containing each optimized piecewise model, primarily for internal use
`summary`	is a dataframe containing breakpoints and fit values for optimal models at each model complexity. If `partitions` is used as a stopping criterion. Other data include: the number of parameters for each model (`k`, determined by the number of breakpoints, independent net-diversification rates, and independent relative-extinction values), the branch or node chosen for each successive piecewise model (`split`), whether the split occurs at a node or stem (`cut`), and the model likelihood (`lnL`)
`FUN`	is a function used to summarize a particular model (indexed by number) and is primarily for internal use

Author(s)

JW Brown <phylo.jwb@gmail.com>, RG FitzJohn, ME Alfaro, LJ Harmon, and JM Eastman

References

Alfaro, ME, F Santini, C Brock, H Alamillo, A Dornburg, DL Rabosky, G Carnevale, and LJ Harmon. 2009. Nine exceptional radiations plus high turnover explain species diversity in jawed vertebrates. *Proceedings of the National Academy of Sciences* **106**: 13410-13414.

See Also

`plot.medusa`

Examples

```
dat=get(data(whales))
phy=dat$phy
richness=dat$richness

## USING AICc as STOPPING CRITERION
res1=medusa(phy, richness, warnings=FALSE)
print(names(res1)) # output list elements
print(res1$summary) # show 'summary' object
summary(res1, criterion="aicc") # select best model based on AICc

## PLOTTING RESULTS
# plot breakpoints for the best model chosen by AICc
# invoking plot.medusa()
plot(res1, cex=0.5,label.offset=1, edge.width=2)
```

name.check *Compares taxa in data and tree*

Description

This function is a general tool for checking for concordance between a data file and a phylogenetic tree. For the data, names can be specified as the names of objects in the vector, rownames of the data array or as 'data.names'. The name.check function finds and lists all taxa present in data set but not in the tree, and vice-versa. The treedata function returns a list containing both the tree and the data after pruning out any species that are not found in both.

Usage

```
name.check(phy, data, data.names=NULL)
```

Arguments

phy	an object of class "phylo"
data	data for tips of the tree
data.names	names of the tips in the order of the data; if this is not given, names will be taken from the names or rownames of the object data

Value

Tree.not.data	Taxa in tree but not data
Data.not.tree	Taxa in data but not tree
...	

Author(s)

Luke J. Harmon

Examples

```
data(geospiza)

tmp <- name.check(geospiza$phy, geospiza$dat)
tmp

## then match data to tree
newphy <- drop.tip(geospiza$phy, tip=tmp$tree_not_data)

## name check should now say "OK"
name.check(newphy, geospiza$dat)

## this can all be done in one step using treedata
td <- treedata(geospiza$phy, geospiza$dat)
```

```
td

all(td$phy$tip.label == newphy$tip.label)
```

`nh.test` *using the Freckleton and Harvey node-height test*

Description

Fits a linear model between the absolute magnitude of the standardized independent contrasts and the height above the root of the node at which they were being compared to identify early bursts of trait evolution.

Usage

```
nh.test(phy, d, regression.type, log = TRUE, rlm.maxit = 20 , show.plot = TRUE, ...)
```

Arguments

`phy`	A time calibrated phylogeny in "phylo" format
`d`	A named vector of trait values.
`regression.type`	The type of regression to be used. Specify `regression.type="lm"` to fit a standard linear model or `regression.type="rlm"` to fit a robust regression model
`log`	Whether the data should be logged or not.
`rlm.maxit`	The maximum number of iterations to fit the robust regression model. This is ignored if `regression.type="lm"`.
`show.plot`	Binary argument indicating whether plot should be made.
`...`	Additional arguments passed to plot

Value

Function returns a `lm` or `rlm` object and outputs a plot if `show.plot=TRUE`

Author(s)

Graham Slater

References

Slater GJ and MW Pennell (in press) Robust regression and posterior predictive simulation increase power to detect early bursts of trait evolution. *Systematic Biology*.

Freckleton RP and PH Harvey (2006) Detecting non-brownian evolution in adaptive radiations. *PLoS Biology* 4:e373.

See Also

pp.mcmc, pic

Examples

```
data(whales)

tmp <- treedata(whales$phy, whales$dat[,1])

phy <- tmp$phy
dat <- tmp$data[,1]

nh.test(phy, dat, regression.type="lm", show.plot=TRUE)
```

---

nodelabel.phylo *Blending information from taxonomies and trees*

Description

working with systematic reference tables and phylogenies

Usage

```
nodelabel.phylo(phy, taxonomy, strict=TRUE, ncores=NULL)
phylo.lookup(taxonomy, ncores=NULL)
lookup.phylo(phy, taxonomy = NULL, clades = NULL, ncores=NULL)
phylo.clades(clades, phy=NULL, unplaced=TRUE, ncores=NULL)
glomogram.phylo(phy, subtrees)
```

Arguments

phy	a phylogenetic tree of class 'phylo' ('multiPhylo' in phylo.clades)
taxonomy	a linkage table (of class matrix) between tips of the phylogeny and clades represented in the tree; rownames of 'taxonomy' should be tips found in the phylogeny
clades	a named list of clade definitions (i.e., spanning taxa; see **Examples**); spanning taxa may invoke other definitions found within the clades list
unplaced	whether to use 'unplaced' taxa if given as an element in clades
subtrees	a list of trees to be grafted into phy; each subtrees element must either be a 'phylo' or 'multiPhylo' object with branch lengths in the same units as phy (see **Examples**)
strict	whether to enforce strict labeling of nodes or allow liberal estimates of the best location of a given nodelabel
ncores	the maximum number of cores to be used

Details

nodelabel.phylo provides a function (as part of the phylo object returned) to resolve the hash key and node identifier for a label found in taxonomy. This function is the FUN element of the returned object. If the taxonomic label cannot be properly placed in the tree (i.e., no subtree is found that is absolutely consistent with the supplied taxonomy, the nearest matching node(s) will be returned when invoking FUN).

phylo.lookup converts a taxonomy into a phylogenetic tree.

lookup.phylo converts a phylogenetic tree (phy) into a linkage table based on nodelabels associated with phy, which can be supplemented with a taxonomy and (or) clades object.

phylo.clades returns a series of phylogenetic subtrees based on clade definitions found in the clades object. Definitions can be handles that are recursive (see **Examples**).

Author(s)

JM Eastman

Examples

```
## Not run:
sal=get(data(caudata))
print(head(sal$tax))

## TREE from TABLE: phylo.lookup()
tax=cbind(sal$tax[,c("subfamily", "family", "suborder")], order="Caudata")
tphy=phylo.lookup(tax, ncores=2)
print(tphy)
head(tphy$node.label)

## TABLE from TREE: lookup.phylo()
tax=sal$tax[,c("genus", "family")]
cld=list(
 Sirenoidea=c("Siren", "Pseudobranchus"),
 Salamandroidea=c("Ambystomatidae", "Plethodontidae"),
 Cryptobranchoidea=c("Hynobius_naevius", "Cryptobranchus_alleganiensis"),
 CAUDATA=c("Sirenoidea","Salamandroidea","Cryptobranchoidea")
)
lkp=lookup.phylo(sal$phy, taxonomy=tax, clades=cld, ncores=2)
print(lkp)
nphy=nodelabel.phylo(sal$phy, lkp, ncores=2)
dev.new()
plot.phylo(ladderize(nphy,right=FALSE), cex=0.35,
type="fan", label.offset=2.5, no.margin=TRUE, edge.color="gray", edge.width=0.5)

nodelabels(nphy$node.label, cex=0.45, col="red", frame="n")

## CLADES to TREE: phylo.clades()
fmrca=geiger:::.mrca
salamandroidea=extract.clade(nphy, fmrca(c("Ambystomatidae", "Plethodontidae"), nphy))
cryptobranchoidea=extract.clade(nphy, fmrca(c("Cryptobranchidae", "Hynobiidae"), nphy))
siren=extract.clade(nphy, fmrca(c("Siren_lacertina", "Siren_intermedia"), nphy))
```

```
clades=list(
    Sirenoidea=c("Siren", "Pseudobranchus"),
    Caudata=c("Sirenoidea","Salamandroidea","Cryptobranchoidea"),
    AMPHIBIA=c("Caudata","Anura","Gymnophiona")
)

phy=list(Cryptobranchoidea=cryptobranchoidea, Salamandroidea=salamandroidea, Siren=siren)
class(phy)="multiPhylo"

res=phylo.clades(clades, phy, ncores=2)
amph=nodelabel.phylo(res$AMPHIBIA, lkp, ncores=2)
print(amph$FUN("Salamandroidea"))
dev.new()
plot(ladderize(amph, right=FALSE), cex=0.2, label.offset=0.05)
nodelabels(amph$node.label, cex=0.35, col="red", frame="n")


## GLOMOGRAM
sirenidae=extract.clade(nphy, fmrca(c("Siren_lacertina", "Pseudobranchus_axanthus"), nphy))
ambystomatidae=extract.clade(nphy, fmrca(c("Ambystoma_gracile", "Ambystoma_texanum"), nphy))
trees=list(
 Cryptobranchoidea=cryptobranchoidea,
 Sirenidae=sirenidae,
 Ambystomatidae=ambystomatidae
)
class(trees)="multiPhylo"

fam=sal$fam
ftax=unique(sal$tax[,c("family", "suborder")])
rownames(ftax)=unname(ftax[,"family"])
fam=nodelabel.phylo(fam, ftax, ncores=2)
fam$FUN("Salamandroidea")

res=glomogram.phylo(fam, trees)
dev.new()
zz=match(res$tip.label, fam$tip.label)
cc=integer(length(zz))
cc[!is.na(zz)]=1

plot(ladderize(res, right=FALSE), cex=1, label.offset=5, tip.color=ifelse(cc==1, "red", "black"))

## End(Not run)
```

`plot.medusa` *MEDUSA: modeling evolutionary diversification using stepwise AIC*

Description

summarizing piecewise diversification models estimated by MEDUSA

Usage

```
 ## S3 method for class 'medusa'
plot(x, cex = 0.5, time = TRUE, ...)
```

Arguments

x	an object of class medusa for plotting
cex	text size
time	logical. should a time axis be plotted. default = TRUE.
...	additional arguments to be passed to internal functions

Details

The medusa model returns a raw list object. This function is used to generate a modified edge matrix (see read.tree for details on the edge matrix), giving all relevant information about the estimated diversification process. The returned **z**-matrix includes: the ancestor (anc) and descendant (dec) relationships between nodes of the tree (using **ape** indices); the beginning (t.0) and ending (t.1) times and length (t.len) of each branch; the diversities at the start (n.0) and end (n.t) of each branch; the piecewise model assigned partition to the branch; whether the branch is associated with a shift; the timing of the shift (t.shift); the net-diversification rate (r) and relative-extinction rate (epsilon) associated with the branch as well as for the direct ancestor of the branch (ancestral.r and ancestral.epsilon). The **z**-matrix also includes a summary attribute that shows which model is chosen and associated information on model fit (see **Examples**).

The raw output of medusa contains an optimized model. The summary output may then be sent to a plotting function which will display the location on the tree where breakpoints have been placed. Note that the first piecewise model corresponds to the root and all descendants (until another breakpoint is encountered).

Author(s)

JW Brown <phylo.jwb@gmail.com>, RG FitzJohn, ME Alfaro, LJ Harmon, and JM Eastman

References

Alfaro, ME, F Santini, C Brock, H Alamillo, A Dornburg, DL Rabosky, G Carnevale, and LJ Harmon. 2009. Nine exceptional radiations plus high turnover explain species diversity in jawed vertebrates. *Proceedings of the National Academy of Sciences* **106**: 13410-13414.

See Also

medusa

Examples

```
dat=get(data(whales))
phy=dat$phy
richness=dat$richness
```

```
res <- medusa(phy, richness)

# select best model based on AICc (showing the third model as best)
plot(res, cex=0.5, label.offset=1) # using plot.medusa()
title("AICc-chosen model")
```

---

pp.mcmc *using posterior predictive MCMC for modeling quantitative trait evolution*

Description

performs posterior predictive checks for models of quantitative trait evolution. At present, only BM, EB, and clade.shift models are implemented

Usage

```
pp.mcmc(phy, d, Ngens = 1000000, sampleFreq = 1000, printFreq = 1000,
        prop.width = 1, model = "BM", eb.type = "exponential",
        clade = NULL, rlm.maxit = 20)
```

Arguments

phy	A time calibrated phylogeny in "phylo" format
d	A named vector or dataframe of trait values.
Ngens	Number of generations that the posterior predictive MCMC will run for. Default is 1 million generations
sampleFreq	The frequency with which model parameters will be sampled from the chain and simulations run. Default is every 1000 generations
printFreq	The frequency with which the current number of generations and acceptance rates will be printed to screen. Default is every 1000 generations
prop.width	The width of the sliding window proposal distribution for ln(Sigmasq) and, if applicable, the exponential change parameter for EB. The width for the EB parameter is obtained by dividing by 10. Default proposal width is 1.
model	The model to fit and simulate under. Default is Brownian motion (BM). Other options are early burst (EB) or an edge shift model (edge.shift) where the rate is allowed to change along an internal edge leading to a specified clade (see argument "clade" and Slater and Pennell in press for an example)
eb.type	The type of exponential change model assumed. If eb.type = "exponential" (the default), then an exponentially declining rate will be assumed and contrasts will be log transformed when computing the node height test. If eb.type = "linear", a linear decline in rate will be assumed and untransformed contrasts will be used.

clade	Default = NULL and is used if model = "BM" or model = "EB". If using model = "edge.shift", then a clade must be specified for which the stem lineage experiences a different rate of evolution. The clade is specified by giving the names of two taxa spanning the clade of interest, e.g. clade = c("A", "B")
rlm.maxit	Maximum number of interations to use for the iteratively reweighted least squares optimization of the robust regression algorithm (see ?rlm). Default is 20 and should be sufficient for most problems

Details

This function runs a posterior predictive MCMC under the specified model, sampling model parameters from their posterior distributions and simulating under that model. Simulated data are summarized using the Node height test (Freckleton and Harvey 2006) slope (OLS and robust regression) and Morphological Disparity Index (Harmon et al. 2003). Model adequacy can then be assessed by comparing observed values for these summary statistics to the posterior predictive distributions

Value

A dataframe containing the following columns:

$generation	the generation at which parameters where sampled and simulations conducted
$logLk	The sampled logLikelihood values for the model
$Sigma	Brownian rate parameter values
$node.height.slope.lm	posterior predictive distribution of slopes for the node height test using an ordinary least squares regression
$node.height.slope.rlm	posterior predictive distribution of slopes for the node height test using a robust regression
$MDI	posterior predictive distribution of MDI values

Author(s)

Graham Slater and Matthew Pennell

References

Slater GJ and MW Pennell (2014) Robust regression and posterior predictive simulation increase power to detect early bursts of trait evolution. *Systematic Biology*.

Freckleton RP and PH Harvey (2006) Detecting non-brownian evolution in adaptive radiations. *PLoS Biology* 4:e373.

Harmon LJ, JA Schulte, A Larson, and JB Losos (2003). Tempo and mode of evolutionary readiations in iguanian lizards. *Science* 301:961-964.

See Also

nh.test, dtt, fitContinuous

Examples

```
data(whales)

tmp <- treedata(whales$phy, whales$dat[,1])

phy <- tmp$phy
dat <- tmp$data[,1]

## compute observed statistics

nht.ols <- nh.test(phy, dat, regression.type = "lm",
log = TRUE, show.plot = FALSE)$coefficients[2,1]

nht.rlm <- nh.test(phy, dat, regression.type = "rlm",
 log = TRUE, show.plot = FALSE)$coefficients[2,1]

mdi.exp <- 0

#---- run short pp.mcmc

pp.eb <- pp.mcmc(phy, dat, Ngens = 1000, sampleFreq = 10, printFreq = 100, model ="EB")

# ---- plot results

# quartz(width = 5, height = 7)
par(mar = c(4,5,1,1))
par(mfcol = c(3,1))


hist(pp.eb$MDI, col = "gray", border = "gray", main = NULL, xlab = "pp.MDI",
ylab = "Frequency", cex.axis = 1.2)

abline(v = mdi.exp, col = "black", lwd = 3, lty = 2)

mdi.p <- length(which(pp.eb$MDI<=0))/length(pp.eb$MDI)

hist(pp.eb$node.height.slope.lm, col = "gray", border = "gray", main = NULL, xlab = "pp.nht_ols",
ylab = "Frequency", cex.axis = 1.2)

abline(v = nht.ols, col = "black", lwd = 3, lty = 2)

node.height.ols.p <- length(which(pp.eb$node.height.slope.lm <= nht.ols)) /
(length(pp.eb$node.height.slope.lm) +1)


hist(pp.eb$node.height.slope.rlm, col = "gray", border = "gray", main = NULL, xlab = "pp.nht_ols",
ylab = "Frequency", cex.axis = 1.2)

abline(v = nht.rlm, col = "black", lwd = 3, lty = 2)

node.height.rr.p <- length(which(pp.eb$node.height.slope.rlm <= nht.rlm)) /
(length(pp.eb$node.height.slope.rlm) +1)
```

r8s.phylo *call r8s from geiger*

Description

call r8s, including a calibration file

Usage

```
r8s.phylo(phy, calibrations=NULL, base="r8srun", ez.run="none", rm=TRUE,
  blformat=c(lengths="persite", nsites=10000, ultrametric="no", round="yes"),
  divtime=c(method="NPRS", algorithm="POWELL"),
  cv=c(cvStart=0, cvInc=0.5, cvNum=8), do.cv=FALSE)
```

Arguments

phy	a phylogram to turn into a chronogram
calibrations	a set of calibrations
base	file name
ez.run	if set to "PL" or "NPRS", does the analysis with settings appropriate for each
rm	remove the output files
blformat	blformat options for r8s. Pay special attention to nsites.
divtime	divtime options for r8s
cv	cv options for r8s
do.cv	Boolean for whether to do cross validation or not

Details

This function uses r8s and a calibration to make your tree ultrametric

Author(s)

JM Eastman & B O'Meara

References

SANDERSON r8s ____NEED TO ADD_____

Examples

```
## Not run:
phy <- read.tree(text=paste0("(Marchantia:0.033817,",
  "(Lycopodium:0.040281,((Equisetum:0.048533",
  "Osmunda:0.033640,Asplenium:0.036526):0.000425):",
  "0.011806,((((Cycas:0.009460,Zamia:0.018847):",
  "0.005021,Ginkgo:0.014702):1.687e-86,((Pinus:",
  "0.021500,(Podocarpac:0.015649,Taxus:0.021081):",
  "0.006473):0.002448,(Ephedra:0.029965,(Welwitsch",
  ":0.011298,Gnetum:0.014165):0.006883):0.016663)",
  ":0.006309):0.010855,((Nymphaea:0.016835,(((((Saururus:",
  "0.019902,Chloranth:0.020151):1.687e-86,",
  "((Araceae:0.020003,(Palmae:0.006005,Oryza:0.031555):",
  "0.002933):0.007654,Acorus:0.038488):0.007844)",
  ":1.777e-83,(Calycanth:0.013524,Lauraceae:0.035902):",
  "0.004656):1.687e-86,((Magnolia:0.015119,Drimys:",
  "0.010172):0.005117,(Ranunculus:0.029027,((Nelumbo:",
  "0.006180,Platanus:0.002347):0.003958,(Buxaceae:",
  "0.013294,((Pisum:0.035675,(Fagus:0.009848,Carya:",
  "0.008236):0.001459):0.001994,(Ericaceae:0.019136,",
  "Solanaceae:0.041396):0.002619):1.687e-86):0.004803)",
  ":1.687e-86):0.006457):0.002918):0.007348,",
  "Austrobail:0.019265):1.687e-86):1.687e-86,Amborella:",
  "0.019263):0.003527):0.021625):0.012469):",
  "0.019372);"))

 calibrations <- data.frame(MRCA="LP", MaxAge=450, MinAge=450,
  taxonA="marchantia", taxonB="pisum", stringsAsFactors=FALSE)

 phy.nprs <- r8s.phylo(phy=phy, calibrations=calibrations, base="nprs_file", ez.run="NPRS")
 phy.pl <- r8s.phylo(phy=phy, calibrations=calibrations, base="pl_file", ez.run="PL")

## End(Not run)
```

`ratematrix` *evolutionary VCV matrix*

Description

estimating the evolutionary or phylogenetic variance-covariance matrix

Usage

```
ratematrix(phy, dat)
```

Arguments

`phy`	a phylogenetic tree of class 'phylo'
`dat`	a named vector or matrix of continuous trait values, associated with species in `phy`

Details

If given dat for n quantitative variables, this function returns the estimated evolutionary variance-covariance matrix of the variables under a multivariate Brownian motion model. Note that other evolutionary models may be possible if the tree is first transformed (see rescale.phylo and **Examples**). If you have n characters in your analysis, this will be an nxn matrix. Diagonal elements represent rate estimates for individual characters, while off-diagonal elements represent the estimated covariance between two characters.

Author(s)

LJ Harmon

References

Revell, L. J., L. J. Harmon, R. B. Langerhans, and J. J. Kolbe. 2007. A phylogenetic approach to determining the importance of constraint on phenotypic evolution in the neotropical lizard, *Anolis cristatellus*. *Evolutionary Ecology Research* 9: 261-282.

Examples

```
geo <- get(data(geospiza))

## EVOLUTIONARY VCV
ratematrix(geo$phy, geo$dat)

## EVOLUTIONARY VCV -- assuming speciational model
kphy <- rescale(geo$phy, "kappa", 0)
ratematrix(kphy, geo$dat)
geo <- get(data(geospiza))

## EVOLUTIONARY VCV
ratematrix(geo$phy, geo$dat)

## EVOLUTIONARY VCV -- assuming speciational model
kphy <- rescale(geo$phy, "kappa", 0)
ratematrix(kphy, geo$dat)
```

---

rc *relative cladogenesis test*

Description

conducting the relative cladogenesis test for all slices through a tree

Usage

```
rc(phy, plot=TRUE, ...)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
plot	whether to plot tree with significant branches highlighted
...	arguments passed for plotting (see plot.phylo)

Details

A list of nodes is returned, along with the number of lineages alive just before that node, the maximum number of descendents that any of those lineages has at the present day, a p-value for this observation under the null hypothesis of a birth-death process (that is, given the null, what is the probability that one of these lineages had at least that many descendents), and the p-value after Bonferroni correction (given that a total of n-1 comparisons are made).

If a plot is made, asterisks will mark significantly diverse clades. These asterisks appear just to the right of the MRCA of the diverse clade.

The Bonferroni correction used here is exceedingly conservative for a tree of any reasonable size (and not necessarily recommended, especially given the exploratory nature of this test and the non-independence of the comparisons). Plotting defaults to indicating which nodes are significant without a Bonferroni correction and a P-value of 0.05 as a cutoff (see **Examples** for modifying this behavior).

One will often see significant results "trickle down" nodes in the tree - that is, if one clade is expecially diverse, then one or more of its parent clades will also be diverse. The most parsimonious place to attribute this effect is to the most shallow significant branch - that is, the branch closest to the tips (see Moore et al. 2004).

Value

Table of results with four columns: Number of ancestors, Maximum descendents, p-value, Bonferroni-corrected p-value

Author(s)

LJ Harmon

References

Purvis A, S Nee, and PH Harvey. 1995. *Proc. R. Soc. London Ser. B* 260:329-333.

Moore BR, KMA Chan, and MJ Donoghue. 2004. Detecting diversification rate variation in supertrees. In O.R.P. Bininda-Emonds (ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pp. 487-533. Kluwer Academic, Netherlands:Dordrecht.

Examples

```
geo <- get(data(geospiza))

## WITHOUT BONFERRONI CORRECTION
rc(geo$phy)

## WITH BONFERRONI CORRECTION and ALPHA=0.15
```

```
rc(geo$phy, bonf=TRUE, p.cutoff=0.15)
```

`rescale.phylo` *Rescale object of class* "phylo"

Description

Applying various transformation to the branches of a phylogenetic tree.

Usage

```
## S3 method for class 'phylo'
rescale(x, model = c("BM", "OU", "EB", "nrate", "lrate",
    "trend", "lambda", "kappa", "delta", "white", "depth"), ...)
```

Arguments

`x`	an object of class "phylo"/
`model`	a model used to transform the tree (see **Details**).
`...`	argument(s) to be passed to the transformation function (see **Examples**).

Details

This function takes a tree and returns either a transformed tree if ... is not empty and gives the parameter value(s) for the tree transformation. If ... is left empty, a function is returned to the user that can be efficiently iterated over many parameter values for transformation. The available models are meant to correspond with changing the model of phenotypic evolution for discrete or continuous characters.

Value

A transformation function (or rescaled phylogenetic tree of class 'phylo' (**ape** format) is returned. Possible transforms include the following:

`\bold{BM}`	is the Brownian motion model, which fits a random walk with variance `sigsq`
`\bold{OU}`	is the Ornstein-Uhlenbeck model (Butler and King 2004), which fits a random walk with a central tendency with an attraction strength proportional to the parameter `alpha`. The `OU` model is called the `hansen` model in **ouch**, although the way the parameters are fit is slightly different here. The parameter used for transformation is `alpha`
`\bold{EB}`	is the Early-burst model (Harmon et al. 2010) and also called the `ACDC` model (accelerating-decelerating; Blomberg et al. 2003). Set by the `a` rate parameter, `EB` fits a model where the rate of evolution increases or decreases exponentially through time, under the model r[t] = r[0] * exp(a * t), where `r[0]` is the initial rate, `a` is the rate change parameter, and `t` is time. The parameter used for transformation is `a`.

\bold{nrate} is the multiple-rates model where time slices have independent rates of evolution. The parameters used for transformation are `time` and `rate`, both of which may be vectors. If rates for several time slices are given, the vectors `time` and `rate` must match in length. The `time` argument is expected in relative branching heights (where the root is 0 and the tips of an ultrametric tree terminate at a relative time of 1). Note that the default behavior is for the multirate transformation to rescale the tree to its original height, while preserving the relative rates across the tree. Note also that the default initial relative rate (`sigsq`, for the rootmost time slice) is 1.

\bold{lrate} is the multiple-rates model where local clades have independent rates of evolution. The parameters used for transformation are `node` and `rate`, both of which may be vectors. If rates for several lineages are given, the vectors `node` and `rate` must match in length. The `node` argument is expected to have node identifiers consistent with **ape** labeling (see `read.tree`). This `node` argument defines where breakpoints occur in the tree (at which a transition to the associated relative rate occurs). Note that the stem branch associated with the node is included as part of the defined rate partition. Rates given are relative to a default rootmost partition with a rate scalar (`sigsq`) of 1.

\bold{trend} is a diffusion model with linear trend in rates through time. The parameter used for transformation is `slope`.

\bold{lambda} is one of the Pagel (1999) models that fits the extent to which the phylogeny predicts covariance among trait values for species. The model effectively transforms the tree as follows: values of `lambda` near 0 cause the phylogeny to become more star-like, and a `lambda` value of 1 recovers the `BM` model. The parameter used for transformation is `lambda`.

\bold{kappa} is a punctuational (speciational) model of trait evolution (Pagel 1999), where character divergence is related to the number of speciation events between two species. Note that if there are missing speciation events in the given phylogeny (due to extinction or incomplete sampling), interpretation under the `kappa` model may be difficult. Considered as a tree transformation, the model raises all branch lengths to an estimated power (`kappa`). The parameter used for transformation is `kappa`.

\bold{delta} is a time-dependent model of trait evolution (Pagel 1999). The `delta` model is similar to `ACDC` insofar as the `delta` model fits the relative contributions of early versus late evolution in the tree to the covariance of species trait values. Where `delta` is greater than 1, recent evolution has been relatively fast; if `delta` is less than 1, recent evolution has been comparatively slow. Intrepreted as a tree transformation, the model raises all node depths to an estimated power (`delta`). The parameter used for transformation is `delta`. Note that the default behavior is for the delta transformation to rescale the tree to its original height.

\bold{white} is a `white`-noise (non-phylogenetic) model, which assumes data come from a single normal distribution with no covariance structure among species

\bold{depth} is simply a transformation of the total depth of the tree; stretching the tree has an effect of increasing rates of evolution under Brownian motion (relative to characters evolved on the unstretched tree), and compressing the tree has the opposite effect. The parameter used for transformation is `depth`.

Author(s)

LJ Harmon and JM Eastman

References

Pagel, M. 1999. Inferring the historical patterns of biological evolution. *Nature* 401:877-884.

Butler, M.A. and A.A. King, 2004. Phylogenetic comparative analysis: a modeling approach for adaptive evolution. *American Naturalist* 164:683-695.

Various papers in prep., L. J. Harmon and J. T. Weir.

Examples

```
geo <- get(data(geospiza))

## returning a function
ltrns <- rescale(geo$phy, "lambda")
plot(ltrns(0))
title("lambda: 0.0")

plot(ltrns(0.5))
title("lambda: 0.5")

plot(ltrns(1))
title("lambda: 1")


## transforming the tree
lphy <- rescale(geo$phy, "lambda", 0.5) # transform tree in one fell swoop
plot(lphy)
title("lambda: 0.5")

## multirate tree -- time
rtrns <- rescale(geo$phy, "nrate")
rphy <- rtrns(time=c(0.2, 0.4, 0.6, 0.8), rate=c(2, 4, 8, 16))
plot(rphy)
title("5-rate tree: by time")

## multirate tree -- lineages
mtrns <- rescale(geo$phy, "lrate")
mphy <- mtrns(node=c(25, 20), rate=c(4, 8))
plot(mphy)
title("3-rate tree: by lineages")
```

`rjmcmc.bm` *Bayesian sampling of shifts in trait evolution: relaxed Brownian motion*

Description

Implements reversible-jump Markov chain Monte Carlo sampling for trait evolutionary models

Usage

```
rjmcmc.bm(phy, dat, SE=NA, ngen = 50000, samp = 100,
    type = c("jump-rbm", "rbm", "jump-bm", "bm"), ...)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
dat	a named vector of continuous trait values, associated with each species in phy
SE	a named vector of standard errors for each trait value; applied to all trait values if given a single value
ngen	number of sampling generations
samp	frequency with which Markov samples are retained (e.g., samp=10 retains every tenth sample in the chain)
type	the class of model to use (see **Details**)
...	arguments passed to make.gbm

Details

Implemented is an MCMC sampler for a general model of Brownian motion, which in the full model (type="jump-rbm") allows relaxed local clocks and also a point process of pulses in evolutionary rate along individual branches. Restricted models include global-rate Brownian motion (type="bm"), relaxed-rates Brownian motion (type="rbm"), and models including jumps but a single rate of diffusion across the tree (type="jump-bm").

Where applicable, posterior estimates of shifts between local rates, estimates of the rates themselves, and inferred jumps (or pulses) are provided as output. Estimates are stored as an MCMC-generations-by-branches matrix (see **Examples**), and branches are uniquely labeled by a cryptographic function to ensure comparability amongst trees differing in topology (see digest).

Note that default settings (as the user assumes if nothing is specified in ...) provide absolutely no guarantee of the chain achieving convergence. The user is emphatically encouraged to supply informed arguments for what are the most critical aspects of this MCMC sampler (see make.gbm for more information on permissible modifications to the MCMC sampler). Finding reasonable run parameters will likely require much trial and error. Run diagnosis and inspection of chain mixing is facilitated by the R package **coda** or by the Java application 'Tracer' (https://beast.community/tracer).

In the **Examples** below, do *not* expect such short chains to reach stationarity!

Value

After a run has completed, acceptance rates for the primary proposal mechanisms are printed to the console, along with settings of control parameters for the run (see make.gbm).

Posterior results are written to several files within a base directory, the contents of which are as follows:

\bold{log} is a logfile including the following for each Markov chain: the generations at which samples were retained (state), the min, max, and median rate of the diffusion process across the tree, the number of evolutionary pulses (jumps) along single branches, the variance associated with the jump process (jumpvar), the root state, and the likelihood (lnL) and prior (lnLp) of the model at sampled generations.

\bold{rda} is a compressed R object, which stores branchwise estimates of the jump and diffusion processes. In order to be interpretable, the rda file should be processed by the function load.rjmcmc. The package **coda** can be used within R for run diagnostics on the processed output (see, e.g., heidel.diag and autocorr.

Author(s)

JM Eastman, LJ Harmon, AL Hipp, and JC Uyeda

References

Eastman JM, ME Alfaro, P Joyce, AL Hipp, and LJ Harmon. 2011. A novel comparative method for identifying shifts in the rate of character evolution on trees. *Evolution* 65:3578-3589.

See Also

load.rjmcmc

Examples

```
## GENERATE DATA: jump-diffusion
phy <- ladderize(sim.bdtree(n=200), right=FALSE)
r <- paste(sample(letters,9,replace=TRUE),collapse="")
defpar <- par(no.readonly=TRUE)


tmp <- ex.jumpsimulator(phy, jumps=10)
dat <- tmp$dat
hist <- tmp$hist

ex.traitgram(phy, hist, alpha=0) # plot history of trait change

## RUN ANALYSIS


## coda package is not a dependency of geiger
## but is very useful for evaluating mcmc runs
## library(coda)


rjmcmc.bm(phy,dat, prop.width=1.5, ngen=20000, samp=500, filebase=r,
          simple.start=TRUE, type="jump-bm")
outdir <- paste("jump-BM", r, sep=".")
ps <- load.rjmcmc(outdir)
```

```
dev.new()
plot(x=ps, par="jumps", burnin=0.25, legend=FALSE, show.tip=FALSE, type="fan", edge.width=2)
mm=match(phy$edge[,2],hist$descendant)
hist=hist[mm,]
edgelabels.auteur(text=NULL, pch=21, cex=hist$cex, bg=NA, col=ifelse(hist$cex>0, 1, NA), lty=2)
title("red (estimated); black (true jump size)", line=-5)
par(defpar)

dev.new()
## from the coda package
coda::autocorr.plot(ps$log, ask=dev.interactive())
plot(ps$log, ask=dev.interactive())

## GENERATE DATA: multi-rate diffusion
scl <- ex.ratesimulator(phy, min=12, show.tip=FALSE)
dat <- rTraitCont(scl)

## RUN ANALYSIS
rjmcmc.bm(phy, dat, prop.width=1.5, ngen=20000, samp=500, filebase=r, simple.start=TRUE, type="rbm")
outdir <- paste("relaxedBM", r, sep=".")
ps <- load.rjmcmc(outdir)
dev.new()
plot(x=ps, par="shifts", burnin=0.25, legend=TRUE, show.tip=FALSE, edge.width=2)

if(!interactive()) { ## clean up
    dirs <- dir(pattern="^(jump-BM|relaxedBM)")
    unlink(dirs, recursive=TRUE)
}
```

`sim.bd` *birth-death population simulator*

Description

simulating species richness (or population growth) under a uniform, time-homogeneous birth-death process

Usage

```
sim.bd(b=1, d=0, n0=1, times=0:4, seed=0)
```

Arguments

`b`	per-lineage birth (speciation) rate
`d`	per-lineage death (extinction) rate
`n0`	number of taxa at starting time zero
`times`	vector of times where extant species are counted
`seed`	random number seed (default is to seed based on the clock)

Details

This function simulates species diversification under a uniform birth-death process. This differs from sim.bdtree in that only the number of species, and not their phylogenetic affinities, are stored. This function relates to bd.ms and bd.km, which are also non-phylogenetic.

Value

a matrix of population size at each time point

Author(s)

RE Glor and LJ Harmon

References

Yule, GU. 1924. A mathematical theory of evolution based on the conclusions of Dr. J. C. Willis, FRS. *Philos. Trans. R. Soc. London Ser. B* 213:21-87

See Also

bd.ms; bd.km

Examples

```
pop1 <- sim.bd(b=0.1, d=0, n0=10, times=1:10)
pop2 <- sim.bd(b=0, d=0.1, n0=10, times=1:10)
pop3 <- sim.bd(b=0.1, d=0.1, n0=10, times=1:10)

plot(pop1, type="l", ylim=c(0,max(c(pop1[,"n"], pop2[,"n"], pop3[,"n"]))))
lines(pop2, col="red")
lines(pop3, col="blue")
```

---

sim.bdtree *birth-death tree simulator*

Description

simulating phylogenetic trees under a uniform birth-death process

Usage

```
sim.bdtree(b=1, d=0, stop=c("taxa", "time"), n=100, t=4, seed=0, extinct=TRUE)
```

Arguments

b	per-lineage birth (speciation) rate
d	per-lineage death (extinction) rate
stop	stopping criterion
n	maximum number of taxa in simulation
t	maximum time steps of simulation
seed	random number seed (default is to seed based on the clock)
extinct	whether to return trees where all lineages have gone extinct (see **Details**)

Details

Starting from a root node - i.e., two living lineages - this function simulates the growth of a phylogenetic tree under a uniform, time-homogeneous birth-death process. This means that every lineage has a constant probability of speciating, and a constant probability of going extinct, per unit time. If birth is greater than death, then the number of lineages is expected to grow exponentially. If `extinct=FALSE`, the function will build trees until one is simulated with at least one surviving lineage.

Value

A phylogenetic tree in 'phylo' format is returned. If death rate is non-zero, then the returned tree will likely include some extinct lineages (terminating before the present day). See `drop.extinct` for a function to remove these lineages.

Note

One note of caution: it is easy to set parameter values that result in tremendously *huge* trees. If the function seems to hang up, this could be the problem.

Other tree simulators are available from the packages **ape** (`rbdtree`), **TreeSim** (`sim.bd.taxa`, `sim.bd.age`, and `sim.bd.taxa.age`), and **phytools** (`pbtree`).

Author(s)

LJ Harmon and J Weir

See Also

`sim.bd` for non-phylogenetic simulations; `drop.extinct`

Examples

```
# Pure-birth tree
p1 <- sim.bdtree(b=0.1, d=0, stop="time", t=20)
plot(p1)

# Birth-death tree with extinct taxa
# The extinct flag prevents trees with no survivors
```

```
p2 <- sim.bdtree(b=0.2, d=0.05, stop="time", t=20, extinct=FALSE)
plot(p2)

# Previous tree with extinct taxa removed
p3 <- drop.extinct(p2)
```

`sim.char` *simulate character evolution*

Description

simulating evolution of discrete or continuous characters on a phylogenetic tree

Usage

```
sim.char(phy, par, nsim = 1, model = c("BM", "speciational", "discrete"), root = 1)
```

Arguments

`phy`	a phylogenetic tree of class 'phylo'
`par`	matrix describing model (either vcv matrix or Q matrix)
`nsim`	number of simulations to run
`model`	a model from which to simulate data
`root`	starting state (value) at root

Details

This function simulates either discrete or continuous data on a phylogenetic tree. The model variable determines the type of simulation to be run. There are three options: `discrete`, which evolves characters under a continuous time Markov model, and two continuous models, `BM` and `speciational`. The `BM` model is a constant rate Brownian-motion model, while `speciational` is a Brownian model on a tree where all branches have the same length. The `model.matrix` parameter gives the structure of the model, and should be either a transition matrix, Q, for the `discrete` model, or a trait variance-covariance matrix for `BM` or `speciational` models. For discrete models, multiple characters may be simulated if `model.matrix` is given as a list of Q matrices (see **Examples**). For continuous models, multivariate characters can be simulated, with their evolution goverened by a covariance matrix specified in the `model.matrix`.

Value

An array of simulated data, either two or three-dimensional, is returned. The first dimension is the number of taxa, the second the number of characters, and the third the number of simulated data sets.

Author(s)

LJ Harmon

Examples

```
## Not run:
geo <- get(data(geospiza))

## Continuous character -- univariate
usims <- sim.char(geo$phy, 0.02, 100)

## Use a simulated dataset in fitContinuous()
fitC <- fitContinuous(geo$phy, usims[,,1], model="BM", control=list(niter=10), ncores=2)

## Continuous character -- multivariate
s <- ratematrix(geo$phy, geo$dat)
csims <- sim.char(geo$phy, s, 100)

## Discrete character -- univariate
q <- list(rbind(c(-.5, .5), c(.5, -.5)))
dsims <- sim.char(geo$phy, q, model="discrete", n=10)

## Use a simulated dataset in fitDiscrete()
fitD <- fitDiscrete(geo$phy, dsims[,,1], model="ER", niter=10, ncores=2)

## Discrete character -- multivariate
qq <- list(rbind(c(-.5, .5), c(.5, -.5)), rbind(c(-.05, .05), c(.05, -.05)))
msims <- sim.char(geo$phy, qq, model="discrete", n=10)

## End(Not run)
```

`startingpt.mecca` *starting values for MECCA*

Description

This function takes the output of calibrateMecca along with observed data and partial least squares loadings and outputs starting values and tuning parameters for the ABC-MCMC (Wegmann et al. 2009).

Usage

```
startingpt.mecca(calibrationOutput, phy, cladeMean, cladeVariance,
                 tolerance = 0.01, plsComponents, BoxCox = TRUE)
```

Arguments

`calibrationOutput`	The output from calibrateMecca
`phy`	A time calibrated phylogeny in "phylo" format
`cladeMean`	A named vector of trait means. All tips in the tree must be represented

cladeVariance	A names vector of trait variances. All tips in the tree must be represented. If only one taxon is present, use 0 for the variance
tolerance	The proportion of calibrations simulations that fall closest to the oberved data that will be retained to compute MECCA tuning parameters
plsComponents	a matrix of Partial Least Squares component loadings
BoxCox	Logical - Should Summary Statistics be Box-Cox transformed? Default is yes and is recommended

Details

You will need to compute PLS loadings using the package "pls" prior to running this function. MECCA performs extremely poorly if summaries are not PLS transformed. If Bayes Factors are to be computed to perform model selection (Leuenberger and Wegmann 2010), raw summaries will need to be used in the post-sampling adjustment step. However, PLS transformed summaries can still be used in the acceptance/rejection step of the MCMC and can also be used to determine which simulations to retain

Value

$tuning	a matrix containing starting values and proposal widths for trait evolution parameters
$startingBirth	Starting value for speciation
$startingDeath	Starting value for extinction
$dcrit	the criticial distance - simulated data must fall within this distance of the observed data in order to be accepted
$obsTraits	the observed data
$plsObserved	Partial Least Squared transformed observed data
$plsLoadings	Partial Least Squared componenent loadings - these are used to transform data during the MCMC

Author(s)

Graham Slater, Luke Harmon, Daniel Wegman

References

Slater GJ, Harmon LJ, Wegmann D, Joyce P, Revell LJ, Alfaro ME. in press Evolution, Leuenberger C, and Wegmann D. 2010. Genetics 184: 243-252., Wegmann D, Leuenberger C, Excoffier L. 2009. Genetics 182: 1207-1218.

Examples

```
example(mecca)
```

subset.phylo *blending information from taxonomies and trees*

Description

working with systematic reference tables and phylogenies

Usage

```
    ## S3 method for class 'phylo'
subset(x, taxonomy, rank="", ncores=1, ...)
```

Arguments

x	a phylogenetic tree of class 'phylo'
taxonomy	a ('matrix') linkage table between tips of the phylogeny and clades represented in the tree; rownames of 'taxonomy' should be tips found in the phylogeny
rank	a column name in 'taxonomy' at which to resample the tree (see **Examples**)
ncores	max number of cores to use
...	arguments to be passed to other functions (has no effect in the present context)

Author(s)

JM Eastman

Examples

```
## Not run:
sal <- get(data(caudata))
print(head(sal$tax))

nphy <- subset(sal$phy, sal$tax, "genus", ncores=1)
plot(nphy, type="fan", cex=0.15)

## End(Not run)
```

tips *descendents of a given node in a phylogenetic tree*

Description

finding descendants of a node in a tree

Usage

```
tips(phy, node)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
node	numeric node-identifier (an integer)

Value

The function returns the set of tips subtended by the given node.

Author(s)

LJ Harmon

Examples

```
geo <- get(data(geospiza))

tips(geo$phy, 18)
```

---

to.auteur *conversion of MCMC samples between auteur and coda*

Description

converting MCMC samples between auteur and coda

Usage

```
to.auteur(obj, phy = NULL, ...)
to.coda(obj)
```

Arguments

obj	for to.auteur – an object of class codaMCMCMC, mcmc.list, or an object exported to an rda file by **auteur**; for to.coda – an object of class auteurMCMCMC or a list of objects individually of class auteurMCMC
phy	a phylogenetic tree of class 'phylo' against which to compile results; if NULL, the tree stored within the rda file is used
...	arguments (burnin and thin) to be passed to load.rjmcmc

Details

A coda format of run(s) is recommended for diagnostic purposes; for summarization, auteur formats are advised. For single chains, the format adopted by both **auteur** and **coda** is identical (an object of class mcmc). For a series of combined runs, formats differ between the **auteur** and **coda** packages: **auteur** requires an intercalated (single) matrix of values, whereas functions within **coda** expect the values to be concatenated into a list (of class mcmc.list). The function to.coda is used solely for pooling multiple runs into a format compatible with the **coda** package.

Value

For to.auteur, an object of class auteurMCMCMC (given multiple runs) or auteurMCMC (given a single run) is returned; for to.coda, an object of class codaMCMCMC is returned.

Author(s)

JM Eastman

See Also

load.rjmcmc

---

treedata *compare taxa in data and tree*

Description

matching species found in a comparative dataset

Usage

```
treedata(phy, data, sort=FALSE, warnings=TRUE)
```

Arguments

phy	a phylogenetic tree of class 'phylo'
data	a named vector or matrix of continuous trait values, for species in phy
sort	whether to sort the data based on names found in phy
warnings	whether to report warnings of mismatched taxa between phy and data

Details

This function is a general tool for checking for concordance between a data object and a phylogenetic tree. For the data, names can be specified as the names of objects in the vector or rownames of the data array.

Value

The function returns a list of two elements (phy and data) that are manipulated to include only those species found in both the tree and data supplied by the user.

Author(s)

LJ Harmon

Examples

```
geo <- get(data(geospiza))

treedata(geo$phy, geo$dat, sort=TRUE, warnings=TRUE)
```

Index

∗ **arith**
bd.ms, 6
dcount, 12
dtt, 14
fitDiscrete, 24
name.check, 42
ratematrix, 52
rc, 53
sim.bd, 60
sim.bdtree, 61
sim.char, 63
∗ **datasets**
geiger-data, 29
∗ **data**
calibrate.rjmcmc, 10
gbresolve, 27
load.rjmcmc, 32
nodelabel.phylo, 44
sim.char, 63
to.auteur, 67
treedata, 68
∗ **graphics**
plot.medusa, 46
∗ **graphs**
congruify.phylo, 11
drop.extinct, 13
gbresolve, 27
nodelabel.phylo, 44
r8s.phylo, 51
rescale.phylo, 55
sim.bdtree, 61
subset.phylo, 66
tips, 66
treedata, 68
∗ **manip**
congruify.phylo, 11
drop.extinct, 13
load.rjmcmc, 32
nodelabel.phylo, 44
r8s.phylo, 51
rescale.phylo, 55
subset.phylo, 66
to.auteur, 67
treedata, 68
∗ **models**
fitContinuous, 16
fitDiscrete, 24
make.gbm, 34
medusa, 39
rjmcmc.bm, 57
∗ **multivariate**
aov.phylo, 5
∗ **univar**
aov.phylo, 5

aicm, 3
aicw, 4
amphibia *(geiger-data)*, 29
anova, *6*
aov, *6*
aov.phylo, 5, *31*
area.between.curves *(geiger-defunct)*, 31
as.Qmatrix.gfit *(fitDiscrete)*, 24
autocorr, *59*

bd.km, *32*, *61*
bd.km *(bd.ms)*, 6
bd.ms, 6, *32*, *61*
BDsim *(geiger-defunct)*, 31
birthdeath.tree *(geiger-defunct)*, 31

calibrate.mecca, 8
calibrate.proposalwidth
(geiger-defunct), 31
calibrate.rjmcmc, 10, *31*
caniformia *(geiger-data)*, 29
carnivores *(geiger-data)*, 29
caudata *(geiger-data)*, 29
chelonia *(geiger-data)*, 29

compare.rates (geiger-defunct), 31
congruify.phylo, 11, *29*
crown.limits (bd.ms), 6
crown.p (bd.ms), 6

dcount, 12, *34*
deltaTree (geiger-defunct), 31
digest, *33*, *58*
disp.calc (geiger-defunct), 31
disparity, *31, 32*
disparity (dtt), 14
drop.extinct, 13, *32*, *62*
drop.random, *32*
drop.random (drop.extinct), 13
drop.tip, *14*
dtt, 14, *31*, *49*
dtt.full (geiger-defunct), 31

ex.jumpsimulator (geiger-example), 32
ex.ratesimulator (geiger-example), 32
ex.traitgram (geiger-example), 32
exponentialchangeTree (geiger-defunct), 31

fastAnc, *31*
fitContinuous, 16, *49*
fitContinuousMCMC, 21
fitDiscrete, 24

gbcontain (gbresolve), 27
gbresolve, 27
geiger (geiger-package), 3
geiger-data, 29
geiger-defunct, 31
geiger-example, 32
geiger-package, 3
geospiza (geiger-data), 29
get.simulation.matrix (geiger-defunct), 31
getAncStates (geiger-defunct), 31
glomogram.phylo (nodelabel.phylo), 44

heidel.diag, *59*

ic.sigma (geiger-defunct), 31
intercalate.samples (geiger-defunct), 31
is.extinct (drop.extinct), 13

kappaTree (geiger-defunct), 31

lambdaTree (geiger-defunct), 31
linearchangeTree (geiger-defunct), 31
lm, *43*
load, *31*
load.rjmcmc, 32, *59*, *67, 68*
lookup.phylo (nodelabel.phylo), 44

make.gbm, *10*, *12*, 34, *58*
mclapply, *19*, *26*
mecca, 36
medusa, *29*, *32*, 39, *47*

name.check, 42
ncbit, *28*
nh.test, 43, *49*
node.leaves (geiger-defunct), 31
node.sons (geiger-defunct), 31
nodelabel.phylo, 44

optim, *17, 18*, *24*, *26*
ouTree (geiger-defunct), 31

pbtree, *62*
phy.anova (geiger-defunct), 31
phy.manova (geiger-defunct), 31
phylo.clades (nodelabel.phylo), 44
phylo.lookup (nodelabel.phylo), 44
pic, *44*
plot.mcmc, *32*
plot.medusa, *41*, 46
plot.phylo, *54*
pool.rjmcmcsamples (geiger-defunct), 31
pp.mcmc, *44*, 48
primates (geiger-data), 29
print.medusa (medusa), 39
prune.extinct.taxa (geiger-defunct), 31
prune.random.taxa (geiger-defunct), 31

r8s.phylo, 51
rate.estimate (geiger-defunct), 31
ratematrix, *5, 6*, *31*, 52
rbdtree, *62*
rc, 53
read.tree, *7*, *15*, *29*, *33*, *47*, *56*
rescale.phylo, *31, 32*, *53*, 55
rescaleTree (geiger-defunct), 31
rjmcmc.bm, *10*, *33–36*, 57
rlm, *43*
runMedusa (geiger-defunct), 31

shifts.plot *(geiger-defunct)*, 31
sim.bd, *31*, 60, *62*
sim.bd.age, *62*
sim.bd.taxa, *62*
sim.bd.taxa.age, *62*
sim.bdtree, *31*, 61, *61*
sim.char, 63
speciationalTree *(geiger-defunct)*, 31
startingpt.mecca, *37*, 64
Startup, *19*, *26*
stem.limits *(bd.ms)*, 6
stem.p *(bd.ms)*, 6
subplex, *17, 18*, *24*, *26*
subset.phylo, 66
summary.manova, *5, 6*
Sys.getenv, *11*

tip.disparity *(geiger-defunct)*, 31
tips, *31*, 66
to.auteur, 67
to.coda *(to.auteur)*, 67
tracer *(geiger-defunct)*, 31
transform.phylo *(geiger-defunct)*, 31
treedata, *40*, 68
tworateTree *(geiger-defunct)*, 31

vmat *(geiger-defunct)*, 31

whales *(geiger-data)*, 29