Package ‘paleotree’

April 13, 2016

Type Package

Version 2.7

Title Paleontological and Phylogenetic Analyses of Evolution

Date 2016-04-12

Author David W. Bapst

Depends R (>= 3.0.0), ape (>= 3.2)

Imports phangorn (>= 1.99.12), stats, phytools, graphics, grDevices, methods, utils

ByteCompile TRUE

Maintainer David W. Bapst <dwbapst@gmail.com>

Description Provides tools for transforming, a posteriori time-scaling, and modifying phylogenies containing extinct (i.e. fossil) lineages. In particular, most users are interested in the functions timePaleoPhy(), bin_timePaleoPhy(), cal3TimePaleoPhy() and bin_cal3TimePaleoPhy(), which a posteriori time-scale cladograms of fossil taxa into dated phylogenies. This package also contains a large number of likelihood functions for estimating sampling and diversification rates from different types of data available from the fossil record (e.g. range data, occurrence data, etc). paleotree users can also simulate diversification and sampling in the fossil record using the function simFossilRecord(), which is a detailed simulator for branching birth-death-sampling processes composed of discrete taxonomic units arranged in ancestor-descendant relationships. Users can use simFossilRecord() to simulate diversification in incompletely sampled fossil records, under various models of morphological differentiation (i.e. the various patterns by which morphotaxa originate from one another), and with time-dependent, longevity-dependent and/or diversity-dependent rates of diversification, extinction and sampling. Additional functions allow users to translate simulated ancestor-descendant data from simFossilRecord() into standard time-scaled phylogenies or unscaled cladograms that reflect the relationships among taxon units.

License CC0

URL https://github.com/dwbapst/paleotree ,
http://webpages.sdsmt.edu/~dbapst/

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-04-13 21:35:16

R topics documented:

paleotree-package . . . 3
binTimeData . . . 5
branchClasses . . . 7
cal3TimePaleoPhy . . . 8
cladogeneticTraitCont . . . 17
communityEcology . . . 19
compareTimescaling . . . 22
constrainParPaleo . . . 24
createMrBayesConstraints . . . 29
dateNodes . . . 30
degradeTree . . . 31
depthRainbow . . . 33
divCurveFossilRecordSim . . . 34
DiversityCurves . . . 35
durationFreq . . . 39
equation2function . . . 43
expandTaxonTree . . . 44
footeValues . . . 46
freqRat . . . 48
graptDisparity . . . 52
graptPBDB . . . 56
horizonSampRate . . . 58
inverseSurv . . . 59
kanto . . . 64
macroperforateForam . . . 67
makePBDBtaxonTree . . . 74
minBranchLength . . . 78
minCharChange . . . 80
modelMethods . . . 86
modifyTerminalBranches . . . 90
multiDiv . . . 95
nearestNeighborDist . . . 98
occData2timeList . . . 100
optimPaleo . . . 103
parentChild2taxonTree . . . 104
perCapitaRates . . . 106
perfectParsCharTree . . . 108
plotOccData . . . 110
plotTraitgram . . . 111
pqr2Ps . . . 113

probAnc 115
resolveTreeChar 116
retiolitinae 121
reverseList 123
rootSplit 124
sampleRanges 125
SamplingConv 129
seqTimeList 131
simFossilRecord 133
simFossilRecordMethods 147
SongZhangDicrano 150
taxa2cladogram 152
taxa2phylo 154
taxonSortPBDBocc 156
taxonTable2taxonTree 160
termTaxa 161
testEdgeMat 165
timeLadderTree 167
timeList2fourDate 168
timePaleoPhy 170
timeSliceTree 179
unitLengthTree 180

Index **182**

`paleotree-package` *paleotree: Paleontological and Phylogenetic Analyses of Evolution*

Description

Analyzes, time-scales and simulates phylogenies of extinct/fossil lineages, along with calculation of diversity curves. Also fits likelihood models to estimate sampling rates from stratigraphic ranges.

Details

Package:	paleotree
Type:	Package
License:	CC0

This package contains functions for analyzing sampling rates given ranges of fossil taxa, in both continuous and discrete time, functions for a posteriori time-scaling phylogenies of fossil taxa and functions for simulating the fossil record in both taxic and phylogenetic varieties.

Author(s)

David W. Bapst

Maintainer: David W. Bapst <dwbapst@gmail.com>

References

Bapst, D.W. 2012. paleotree: an R package for paleontological and phylogenetic analyses of evolution. *Methods in Ecology and Evolution*. 3: 803-807. doi: 10.1111/j.2041-210X.2012.00223.x

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Bapst, D. W. 2013. When Can Clades Be Potentially Resolved with Morphology? *PLoS ONE*. 8(4):e62312.

Bapst, D. W. 2014. Assessing the effect of time-scaling methods on phylogeny-based analyses in the fossil record. **Paleobiology 40**(3):331-351.

See Also

This package relies extensively on the phylogenetic toolkit and standards offered by the ape package, and hence lists this package as a depends, so it is loaded simultaneously.

Examples

```
# get the package version of paleotree
packageVersion("paleotree")

# get the citation for paleotree
citation("paleotree")

##Simulate some fossil ranges with simFossilRecord
set.seed(444);
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#let's see what the 'true' diversity curve looks like in this case
#plot the FADs and LADs with taxicDivCont()
taxicDivCont(taxa)

#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.5)
#plot the diversity curve based on the sampled ranges
layout(1:2)
taxicDivCont(rangesCont)
#Now let's use binTimeData to bin in intervals of 10 time units
rangesDisc <- binTimeData(rangesCont,int.length=10)
#plot with taxicDivDisc
taxicDivDisc(rangesDisc)
#compare to the continuous time diversity curve

layout(1)

#taxa2phylo assumes we know speciation events perfectly... what if we don't?
#first, let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
```

```
cladogram <- taxa2cladogram(taxa,plot=TRUE)
#Now let's try timePaleoPhy using the continuous range data
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",plot=TRUE)
#plot diversity curve
phyloDiv(ttree,drop.ZLB=TRUE)

#that tree lacked the terminal parts of ranges (tips stops at the taxon FADs)
#let's add those terminal ranges back on with add.term
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",add.term=TRUE,plot=TRUE)
#plot diversity curve
phyloDiv(ttree)
```

`binTimeData` *Bin Simulated Temporal Ranges in Discrete Intervals*

Description

Converts a matrix of simulated continuous-time first occurrences and last occurrences for fossil taxa into first and last occurrences given in some set of simulated/placed discrete-time intervals, which is output along with information of the dates of the given intervals.

Usage

```
binTimeData(timeData, int.length = 1, start = NA, int.times = NULL)
```

Arguments

`timeData`	Two-column matrix of simulated first and last occurrences in absolute continuous time
`int.length`	Time interval length, default is 1 time unit
`start`	Starting time for calculating the intervals.
`int.times`	A two column matrix with the start and end times of the intervals to be used.

Details

This function takes a simulated matrix of per-taxon first and last occurrences and, by dividing the time-scale into time intervals of non-zero length, lists taxon occurrences within those interval. By default, a set of sequential non-overlapping time-interval of equal non-zero length are used, with the length controlled by the argument int.length.

Alternatively, a two column matrix of interval start and end times to be used can be input via the argument int.times. None of these intervals can have a duration (temporal length) greater than zero. If a first or last appearance in the input range data could fit into multiple intervals (i.e. the input discrete time intervals are overlapping), then the appearance data is placed in the interval of the shortest duration. When output, the interval times matrix (see below) will be sorted from first to last.

This function is SPECIFICALLY for simulating the effect of having a discrete time-scale for analyses using simulations. This function should not be used for non-simulations uses, such as binning temporal occurrences for analyses of real data. In those case, the temporal ranges (which, in real data, will probably be given as discrete time intervals) should already be tabulated within discrete intervals prior to use in paleotree. The user should place the temporal information in a list object, as described for the output of binTimeData (see below).

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero. However, the numbering of intervals giving in the output increases with time, as these are numbered relative to each other, from first to last.

As of version 1.7 of paleotree, taxa which are extant as indicated in timeData as being in a time interval bounded (0,0), unless time-bins are preset using argument int.times (prior to version 1.5 they were erroneously listed as NA).

Value

A list containing:

`int.times`	A 2 column matrix with the start and end times of the intervals used; time decreases relative to the present.
`taxon.times`	A 2 column matrix with the first and last occurrences of taxa in the intervals listed in int.times, with numbers referring to the row of int.times.

Author(s)

David W. Bapst

See Also

`simFossilRecord`, `sampleRanges`, `taxicDivCont`

Examples

```
#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5)
#Now let's use binTimeData() to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length=1)
#plot with taxicDivDisc()
equalDiscInt <- taxicDivDisc(rangesDisc)

#example with pre-set intervals input (including overlapping)
presetIntervals <- cbind(c(1000,990,970,940),c(980,970,950,930))
rangesDisc1 <- binTimeData(rangesCont,int.times=presetIntervals)
taxicDivDisc(rangesDisc1)
#now let's plot diversity with (different) equal length intervals used above
```

```
taxicDivDisc(rangesDisc1,int.times=equalDiscInt[,1:2])

#example with extant taxa
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40))
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5,,modern.samp.prob=1)
#Now let's use binTimeData() to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length=1)
#plot with taxicDivDisc()
taxicDivDisc(rangesDisc)

#example with pre-set intervals input (including overlapping)
presetIntervals <- cbind(c(40,30,20,10),c(30,20,10,0))
rangesDisc1 <- binTimeData(rangesCont,int.times=presetIntervals)
taxicDivDisc(rangesDisc1)
```

---

branchClasses — *Partitions the branch lengths of a tree into several classes based on their placement.*

Description

Partitions the branch lengths of a tree into several classes based on their placement.

Usage

```
branchClasses(tree, whichExtant = NULL, tol = 0.01)
```

Arguments

`tree`	A time-scaled phylogeny to be analysed, as an object of class phylo, ideally with a $root.time element as is typical for paleotree output phylogenies. If $root.time is not present, the most recent tips will be interpreted as being at the modern day (i.e. 0 time-units before present).
`whichExtant`	A logical vector with length equal to number of tips in the tree. TRUE indicates a taxon that is extant at the moden day. If present
`tol`	Tolerance used to distinguish extant taxa, if whichExtant is not provided, to avoid issues with number rounding. Taxa within tol of the modern day will be considered extant.

Details

This function will partition the internode (node to node, including internal node to terminal tip) branch lengths of a tree into four separate classes: all (all the internode branches of a tree), int (internal branches which run from one internode to another), live (terminal branches which run from an internal node to a terminal tip representing an extinction event before the present) and dead (terminal branches which run from an internal node to a terminal tip at the modern day, reflecting a still-living taxon).

The depths of the internal 'mother' node (i.e. time of origin, before the modern day of each branch length are included as the /codenames of the branch length vectors.

This function is mainly of use for modeling internode branch lengths in a phylogeny including fossil taxa.

Value

The output is a list consisting of four vectors, with the /codenames of the vectors being their corresponding time of origin. See details.

Examples

```
#simulated example
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=c(10,20))
taxa<-fossilRecord2fossilTaxa(record)
tree <- taxa2phylo(taxa)
brlenRes <- branchClasses(tree)

#see frequency histograms of branch lengths
layout(1:4)
for(x in 1:length(brlenRes)){
hist(brlenRes[[x]],main="Branch Lengths",xlab=names(brlenRes)[x])
}

#see frequency histograms of branch depths
layout(1:4)
for(x in 1:length(brlenRes)){
hist(as.numeric(names(brlenRes[[x]])),main="Branch Depths",xlab=names(brlenRes)[x])
}

layout(1)
```

`cal3TimePaleoPhy` *Three Rate Calibrated 'a posteriori' Time-Scaling of Paleo-Phylogenies*

Description

Time-scales an unscaled cladogram of fossil taxa, using information on their ranges and estimates of the instantaneous rates of branching, extinction and sampling. The output is a sample of a posteriori time-scaled trees, as resulting from a stochastic algorithm which samples observed gaps in the fossil record with weights calculated based on the input rate estimates. This function also uses the three-rate calibrated time-scaling algorithm to stochastically resolve polytomies and infer potential ancestor-descendant relationships, simultaneous with the time-scaling treatment.

Usage

```
cal3TimePaleoPhy(tree, timeData, brRate, extRate, sampRate, ntrees = 1,
  anc.wt = 1, node.mins = NULL, dateTreatment = "firstLast",
  FAD.only = FALSE, adj.obs.wt = TRUE, root.max = 200, step.size = 0.1,
  randres = FALSE, noisyDrop = TRUE, tolerance = 1e-04,
  diagnosticMode = FALSE, plot = FALSE)

bin_cal3TimePaleoPhy(tree, timeList, brRate, extRate, sampRate, ntrees = 1,
  anc.wt = 1, node.mins = NULL, dateTreatment = "firstLast",
  FAD.only = FALSE, sites = NULL, point.occur = FALSE,
  nonstoch.bin = FALSE, adj.obs.wt = TRUE, root.max = 200,
  step.size = 0.1, randres = FALSE, noisyDrop = TRUE, tolerance = 1e-04,
  diagnosticMode = FALSE, plot = FALSE)
```

Arguments

`tree`	An unscaled cladogram of fossil taxa, of class `phylo`. Tip labels must match the taxon labels in the respective temporal data.
`timeData`	Two-column matrix of first and last occurrences in absolute continuous time, with row names as the taxon IDs used on the tree. This means the first column is very precise FADs (first appearance dates) and the second column is very precise LADs (last appearance dates), reflect the precise points in time when taxa first and last appear. If there is stratigraphic uncertainty in when taxa appear in the fossil record, it is preferable to use the 'bin' time-scaling functions; however, see the argument `dateTreatment`.
`brRate`	Either a single estimate of the instanteous rate of branching (also known as the 'per-capita' origination rate, or speciation rate if taxonomic level of interest is species) or a vector of per-taxon estimates
`extRate`	Either a single estimate of the instanteous extinction rate (also known as the 'per-capita' extinction rate) or a vector of per-taxon estimates
`sampRate`	Either a single estimate of the instanteous sampling rate or a vector of per-taxon estimates
`ntrees`	Number of time-scaled trees to output
`anc.wt`	Weighting against inferring ancestor-descendant relationships. The argument anc.wt allows users to change the default consideration of anc-desc relationships. This value is used as a multiplier applied to the probability of choosing any node position which would infer an ancestor-descendant relationship. By default, anc.wt=1, and thus these probabilities are unaltered. if anc.wt is less

than 1, the probabilities decrease and at anc.wt=0, no ancestor-descendant relationships are inferred at all. Can be a single value or a vector of per-taxon values, such as if a user wants to only allow plesiomorphic taxa to be ancestors.

`node.mins` The minimum dates of internal nodes (clades) on a phylogeny can be set using node.mins. This argument takes a vector of the same length as the number of nodes, with dates given in the same order as nodes are ordered in the `tree$edge` matrix. Note that in `tree$edge`, terminal tips are given the first set of numbers (`1:Ntip(tree)`), so the first element of `node.mins` is the first internal node (the node numbered `Ntip(tree)+1`, which is generally the root for most `phylo` objects read by `read.tree`). Not all nodes need be given minimum dates; those without minimum dates can be given as NA in `node.mins`, but the vector must be the same length as the number of internal nodes in `tree`. These are minimum date constraints, such that a node will be 'frozen' by the cal3 algorithm so that constrained nodes will always be *at least as old as this date*, but the final date may be even older depending on the taxon dates used, the parameters input for the cal3 algorithm and any other minimum node dates given (e.g. if a clade is given a very old minimum date, this will (of course) over-ride any minimum dates given for clades that that node is nested within). if the constrained nodes include a polytomy, this polytomy will still be resolved with respect to the cal3 algorithm, but the first divergence will be 'frozen' so that it is at least as old as the minimum age, while any additional divergences will be allowed to occur after this minimum age.

`dateTreatment` This argument controls the interpretation of timeData. The default setting 'firstLast' treats the dates in timeData as a column of precise first and last appearances. A second option, added by great demand, is 'minMax' which treats these dates as minimum and maximum bounds on single point dates. Under this option, all taxa in the analysis will be treated as being point dates, such that the first appearance is also the last. These dates will be pulled under a uniform distribution. If 'minMax' is used, add.term becomes meaningless, and the use of it will return an error message. A third option is 'randObs'. This assumes that the dates in the matrix are first and last appearance times, but that the desired time of observation is unknown. Thus, this is much like 'firstLast' except the effective time of observation (the taxon's LAD under 'firstLast') is treated an uncertain date, and is randomly sampled between the first and last appearance times. The FAD still is treated as a fixed number, used for dating the nodes. In previous versions of paleotree, this was called in `cal3timePaleoPhy` using the argument rand.obs, which has been removed for clarity. This temporal uncertainty in times of observation might be useful if a user is interested in applying phylogeny-based approaches to studying trait evolution, but have per-taxon measurements of traits that come from museum specimens with uncertain temporal placement. With both arguments 'minMax' and 'randObs', the sampling of dates from random distributions should compel users to produce many time-scaled trees for any given analytical purpose. Note that 'minMax' returns an error in 'bin' time-scaling functions; please use 'points.occur' instead.

`FAD.only` Should the tips represent observation times at the start of the taxon ranges? If TRUE, result is similar to when terminal ranges are no added on with timePaleoPhy. If FAD.only is TRUE and dateTreatment is 'minMax' or 'randObs', the function will stop and a warning will be produced, as these combinations imply

	contradictory sets of times of observation.
`adj.obs.wt`	If the time of observation are before the LAD of a taxon, should the weight of the time of observation be adjusted to account for the known observed history of the taxon which occurs AFTER the time of observation? Should only have an effect if time of observation ISN'T the LAD, if times of observation for a potential ancestor are earlier than the first appearance of their potential descendants and if the ancestral weights for taxa aren't set to zero (so there can be potential ancestors).
`root.max`	Maximum time before the first FAD that the root can be pushed back to.
`step.size`	Step size of increments used in zipper algorithm to assign node ages.
`randres`	Should polytomies be randomly resolved using multi2di in ape rather than using the cal3 algorithm to weight the resolution of polytomies relative to sampling in the fossil record?
`noisyDrop`	If `TRUE` (the default), any taxa dropped from tree due to not having a matching entry in the time data will be listed in a system message.
`tolerance`	Acceptable amount of shift in tip dates from dates listed in `timeData`. Shifts outside of the range of `tolerance` will cause a warning message to be issued that terminal tips appear to be improperly aligned.
`diagnosticMode`	If `TRUE`, `cal3timePaleoPhy` will return to the console and to graphic devices an enormous number of messages, plots and anciliary information that may be useful or entirely useless to figuring out what is going wrong.
`plot`	If true, plots the input, "basic" time-scaled phylogeny (an intermediary step in the algorithm) and the output cal3 time-scaled phylogeny.
`timeList`	A list composed of two matrices giving interval times and taxon appearance dates. The rownames of the second matrix should be the taxon IDs, identical to the `tip.labels` for tree. See details.
`sites`	Optional two column matrix, composed of site IDs for taxon FADs and LADs. The sites argument allows users to constrain the placement of dates by restricting multiple fossil taxa whose FADs or LADs are from the same very temporally restricted sites (such as fossil-rich Lagerstatten) to always have the same date, across many iterations of time-scaled trees. To do this, simply give a matrix where the "site" of each FAD and LAD for every taxon is listed, as corresponding to the second matrix in timeList. If no sites matrix is given (the default), then it is assumed all fossil come from different "sites" and there is no shared temporal structure among the events.
`point.occur`	If true, will automatically produce a 'sites' matrix which forces all FADs and LADs to equal each other. This should be used when all taxa are only known from single 'point occurrences', i.e. each is only recovered from a single bed/horizon, such as a Lagerstatten.
`nonstoch.bin`	If true, dates are not stochastically pulled from uniform distributions. See below for more details.

Details

The three-rate calibrated ("cal3") algorithm time-scales trees a posteriori by stochastically picking node divergence times relative to a probability distribution of expected waiting times between speciation and first appearance in the fossil record. This algorithm is extended to apply to resolving polytomies and designating possible ancestor-descendant relationships. The full details of this method, its sister method the sampling-rate-calibrated time-scaling method and the details of the algorithm will be given in Bapst (in prep for Paleobiology).

Briefly, cal3 time-scaling is done by examining each node separately, moving from the root upwards. Ages of branching nodes are constrained below by the ages of the nodes below them (except the root; hence the need for the root.max argument) and constrained above by the first appearance dates (FADs) of the daughter lineages. The position of the branching event within this constrained range implies different amounts of unobserved evolutionary history. cal3 considers a large number of potential positions for the branching node (the notation in the code uses the analogy of viewing the branching event as a 'zipper') and calculates the summed unobserved evolutionary history implied by each branching time. The probability density of each position is then calculated under a gamma distribution with a shape parameter of 2 (implying that it is roughly the sum of two normal waiting times under an exponential) and a rate parameter which takes into account both the probability of not observing a lineage of a certain duration and the 'twiginess' of the branch, i.e. the probability of having short-lived descendants which went extinct and never were sampled (ala Friedman and Brazeau, 2011). These densities calculated under the gamma distribution are then used as weights to stochastically sample the possible positions for the branching node. This basic framework is extended to polytomies by allowing a branching event to fall across multiple potential lineages, adding each lineage one by one, from earliest appearing to latest appearing (the code notation refers to this as a 'parallel zipper').

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

These functions will intuitively drop taxa from the tree with NA for range or that are missing from timeData.

The sampling rate used by cal3 methods is the instantaneous sampling rate, as estimated by various other function in the paleotree package. See `make_durationFreqCont` for more details. If you have the per-time unit sampling probability ('R' as opposed to 'r') look at the sampling parameter conversion functions also included in this package (e.g. `sProb2sRate`). Most datasets will probably use `make_durationFreqDisc` and `sProb2sRate` prior to using this function, as shown in an example below.

The branching and extinction rate are the 'per-capita' instantaneous origination/extinction rates for the taxic level of the tips of the tree being time-scaled. Any user of the cal3 time-scaling method has multiple options for estimating these rates. One is to separately calculate the per-capita rates (following the equations in Foote, 2001) across multiple intervals and take the mean for each rate. A second, less preferred option, would be to use the extinction rate calculated from the sampling rate above (under ideal conditions, this should be very close to the mean 'per-capita' rate calculated from by-interval FADs and LADs). The branching rate in this case could be assumed to be very close to the extinction rate, given the tight relationship observed in general between these two (Stanley, 1976; see Foote et al., 1999, for a defense of this approach), and thus the extinction rate estimate could be used also for the branching rate estimate. (This is what is done for the examples below.) A third option for calculating all three rates simultaneously would be to apply likelihood methods developed by Foote (2002) to forward and reverse survivorship curves. Note that only one of these three suggested methods is implemented in `paleotree`: estimating the sampling and extinction rates from the distribution of taxon durations via `make_durationFreqCont` and `make_durationFreqDisc`.

By default, the cal3 functions will consider that ancestor-descendant relationships may exist among the given taxa, under a budding cladogenetic or anagenetic modes. Which tips are designated as which is given by two additional elements added to the output tree, $budd.tips (taxa designated as ancestors via budding cladogenesis) and $anag.tips (taxa designated as ancestors via anagenesis). This can be turned off by setting `anc.wt = 0`. As this function may infer anagenetic relationships during time-scaling, this can create zero-length terminal branches in the output. Use `dropZLB` to get rid of these before doing analyses of lineage diversification.

Unlike `timePaleoPhy`, cal3 methods will always resolve polytomies. In general, this is done using the rate calibrated algorithm, although if argument `randres = TRUE`, polytomies will be randomly resolved with uniform probability, ala multi2di from ape. Also, cal3 will always add the terminal ranges of taxa. However, because of the ability to infer potential ancestor-descendant relationships, the length of terminal branches may be shorter than taxon ranges themselves, as budding may have occurred during the range of a morphologically static taxon. By resolving polytomies with the cal3 method, this function allows for taxa to be ancestral to more than one descendant taxon. Thus, users who believe their dataset may contain indirect ancestors are encouraged by the package author to try cal3 methods with their consensus trees, as opposed to using the set of most parsimonious trees. Comparing the results of these two approaches may be very revealing.

Like `timePaleoPhy`, `cal3TimePaleoPhy` is designed for direct application to datasets where taxon first and last appearances are precisely known in continuous time, with no stratigraphic uncertainty. This is an uncommon form of data to have from the fossil record, although not an impossible form (micropaleontologists often have very precise range charts, for example). This means that most users *should not* use `cal3TimePaleoPhy` directly, unless they have written their own code to deal with stratigraphic uncertainty. For some groups, the more typical 'first' and 'last' dates represent the minimum and maximum absolute ages for the fossil collections that a taxon is known is known from. Presumably, the first and last appearances of that taxon in the fossil record is at unknown dates within these bounds. These should not be mistaken as the FADs and LADs desired by `cal3TimePaleoPhy`, as `cal3TimePaleoPhy` will use the earliest dates provided to calibrate node ages, which is either an overly conservative approach to time-scaling or fairly nonsensical.

Alternatively to using `cal3TimePaleoPhy`, `bin_cal3TimePaleoPhy` is a wrapper of `cal3TimePaleoPhy` which produces time-scaled trees for datasets which only have interval data available. For each output tree, taxon first and last appearance dates are placed within their listed intervals under a uniform distribution. Thus, a large sample of time-scaled trees will approximate the uncertainty in the actual timing of the FADs and LADs.

The input `timeList` object can have overlapping (i.e. non-sequential) intervals, and intervals of uneven size. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. If taxa occur only in single collections (i.e. their first and last appearance in the fossil record is synchronous, the argument point.occur will force all taxa to have instantaneous durations in the fossil record. Otherwise, by default, taxa are assumed to first and last appear in the fossil record at different points in time, with some positive duration. The sites matrix can be used to force only a portion of taxa to have simultaneous first and last appearances.

By setting the argument `nonstoch.bin` to `TRUE` in `bin_cal3TimePaleoPhy`, the dates are NOT stochastically pulled from uniform bins but instead FADs are assigned to the earliest time of whichever interval they were placed in and LADs are placed at the most recent time in their placed interval. This option may be useful for plotting. The sites argument becomes arbitrary if nonstoch.bin is `TRUE`.

If `timeData` or the elements of `timeList` are actually data frames (as output by `read.csv` or `read.table`), these will be coerced to a matrix.

A tutorial for applying the time-scaling functions in paleotree, particularly the cal3 method, along with an example using real (graptolite) data, can be found at the following link:

http://nemagraptus.blogspot.com/2013/06/a-tutorial-to-cal3-time-scaling-using.html

Value

The output of these functions is a time-scaled tree or set of time-scaled trees, of either class phylo or multiphylo, depending on the argument ntrees. All trees are output with an element `$root.time`. This is the time of the root on the tree and is important for comparing patterns across trees.

Additional elements are `sampledLogLike` and `$sumLogLike` which respectively record a vector containing the 'log-densities' of the various node-ages selected for each tree by the 'zipper' algorithm, and the sum of those log-densities. Although they are very similar to log-likelihood values, they may not be true likelihoods, as node ages are conditional on the other ages selected by other nodes. However, these values may give an indication about the relative optimality of a set of trees output by the cal3 functions.

Trees created with `bin_cal3TimePaleoPhy` will output with some additional elements, in particular `$ranges.used`, a matrix which records the continuous-time ranges generated for time-scaling each tree. (Essentially a pseudo-timeData matrix.)

Note

Most importantly, please note the stochastic element of the three rate-calibrated time-scaling methods. These do not use traditional optimization methods, but instead draw divergence times from a distribution defined by the probability of intervals of unobserved evolutionary history. This means analyses MUST be done over many cal3 time-scaled trees for analytical rigor! No one tree is correct.

Similarly, please account for stratigraphic uncertainty in your analysis. Unless you have exceptionally resolved data, use a wrapper with the cal3 function, either the provided bin_cal3TimePaleoPhy or code a wrapper function of your own that accounts for stratigraphic uncertainty in your dataset. Remember that the FADs (earliest dates) given to timePaleoPhy will *always* be used to calibrate node ages!

Author(s)

David W. Bapst

References

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Foote, M. 2000. Origination and extinction components of taxonomic diversity: general problems. Pp. 74-102. In D. H. Erwin, and S. L. Wing, eds. Deep Time: Paleobiology's Perspective. The Paleontological Society, Lawrence, Kansas.

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Friedman, M., and M. D. Brazeau. 2011. Sequences, stratigraphy and scenarios: what can we say about the fossil record of the earliest tetrapods? *Proceedings of the Royal Society B: Biological Sciences* 278(1704):432-439.

Stanley, S. M. 1979. Macroevolution: Patterns and Process. W. H. Freeman, Co., San Francisco.

See Also

timePaleoPhy, binTimeData, make_durationFreqCont, sProb2sRate, multi2di

Examples

```
#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot=TRUE)
#this library allows one to use rate calibrated type time-scaling methods (Bapst, in prep.)
#to use these, we need an estimate of the sampling rate (we set it to 0.5 above)
likFun<-make_durationFreqCont(rangesCont)
srRes<-optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
      method="L-BFGS-B",control=list(maxit=1000000))
sRate <- srRes[[1]][2]
# we also need extinction rate and branching rate
   # we can get extRate from getSampRateCont too
#we'll assume extRate=brRate (ala Foote et al., 1999); may not always be a good assumption
divRate<-srRes[[1]][1]
#now let's try cal3TimePaleoPhy, which time-scales using a sampling rate to calibrate
#This can also resolve polytomies based on sampling rates, with some stochastic decisions
ttree <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    sampRate=sRate,ntrees=1,plot=TRUE)
#notice the warning it gives!
phyloDiv(ttree)

#by default, cal3TimePaleoPhy may predict indirect ancestor-descendant relationships
#can turn this off by setting anc.wt=0
ttree <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    sampRate=sRate,ntrees=1,anc.wt=0,plot=TRUE)


#let's look at how three trees generated with very different time of obs. look
ttreeFAD <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    FAD.only=TRUE,dateTreatment="firstLast",sampRate=sRate,ntrees=1,plot=TRUE)
ttreeRand <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    FAD.only=FALSE,dateTreatment="randObs",sampRate=sRate,ntrees=1,plot=TRUE)
#by default the time of observations are the LADs
ttreeLAD <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    FAD.only=FALSE,dateTreatment="randObs",sampRate=sRate,ntrees=1,plot=TRUE)
layout(1:3)
parOrig <- par(no.readonly=TRUE)
par(mar=c(0,0,0,0))
plot(ladderize(ttreeFAD));text(5,5,"time.obs=FAD",cex=1.5,pos=4)
plot(ladderize(ttreeRand));text(5,5,"time.obs=Random",cex=1.5,pos=4)
plot(ladderize(ttreeLAD));text(5,5,"time.obs=LAD",cex=1.5,pos=4)
layout(1); par(parOrig)

#to get a fair sample of trees, let's increase ntrees
ttrees <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=divRate,extRate=divRate,
    sampRate=sRate,ntrees=9,plot=FALSE)
#let's compare nine of them at once in a plot
layout(matrix(1:9,3,3))
parOrig <- par(no.readonly=TRUE)
par(mar=c(0,0,0,0))
for(i in 1:9){plot(ladderize(ttrees[[i]]),show.tip.label=FALSE)}
layout(1)
par(parOrig)
#they are all a bit different!

#can plot the median diversity curve with multiDiv
multiDiv(ttrees)

#using node.mins
#let's say we have (molecular??) evidence that node #5 is at least 1200 time-units ago
#to use node.mins, first need to drop any unshared taxa
droppers <- cladogram$tip.label[is.na(
      match(cladogram$tip.label,names(which(!is.na(rangesCont[,1])))))]
cladoDrop <- drop.tip(cladogram, droppers)
# now make vector same length as number of nodes
nodeDates <- rep(NA, Nnode(cladoDrop))
nodeDates[5]<-1200
ttree <- cal3TimePaleoPhy(cladoDrop,rangesCont,brRate=divRate,extRate=divRate,
    sampRate=sRate,ntrees=1,node.mins=nodeDates,plot=TRUE)

#example with time in discrete intervals
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot=TRUE)
#Now let's use binTimeData to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length=1)
#we can do something very similar for the discrete time data (can be a bit slow)
likFun<-make_durationFreqDisc(rangesDisc)
spRes<-optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
      method="L-BFGS-B",control=list(maxit=1000000))
sProb <- spRes[[1]][2]
#but that's the sampling PROBABILITY per bin, not the instantaneous rate of change
#we can use sProb2sRate() to get the rate. We'll need to also tell it the int.length
sRate1 <- sProb2sRate(sProb,int.length=1)
#we also need extinction rate and branching rate (see above)
    #need to divide by int.length...
divRate<-spRes[[1]][1]/1
```

```
#estimates that r=0.3... kind of low (simulated sampling rate is 0.5)
#Note: for real data, you may need to use an average int.length (no constant length)
ttree <- bin_cal3TimePaleoPhy(cladogram,rangesDisc,brRate=divRate,extRate=divRate,
    sampRate=sRate1,ntrees=1,plot=TRUE)
phyloDiv(ttree)
#can also force the appearance timings not to be chosen stochastically
ttree1 <- bin_cal3TimePaleoPhy(cladogram,rangesDisc,brRate=divRate,extRate=divRate,
    sampRate=sRate1,ntrees=1,nonstoch.bin=TRUE,plot=TRUE)
phyloDiv(ttree1)

# testing node.mins in bin_cal3TimePaleoPhy
ttree <- bin_cal3TimePaleoPhy(cladoDrop,rangesDisc,brRate=divRate,extRate=divRate,
    sampRate=sRate1,ntrees=1,node.mins=nodeDates,plot=TRUE)
# with randres=TRUE
ttree <- bin_cal3TimePaleoPhy(cladoDrop,rangesDisc,brRate=divRate,extRate=divRate,
    sampRate=sRate1,ntrees=1,randres=TRUE,node.mins=nodeDates,plot=TRUE)


#example with multiple values of anc.wt
ancWt <- sample(0:1,nrow(rangesDisc[[2]]),replace=TRUE)
names(ancWt)<-rownames(rangesDisc[[2]])
ttree1 <- bin_cal3TimePaleoPhy(cladogram,rangesDisc,brRate=divRate,extRate=divRate,
    sampRate=sRate1,ntrees=1,anc.wt=ancWt,plot=TRUE)
```

`cladogeneticTraitCont` *Simulate Cladogenetic Trait Evolution*

Description

This function simulates trait evolution at each speciation/branching event in a matrix output from `simFossilRecord`, after transformation with `fossilRecord2fossilTaxa`.

Usage

```
cladogeneticTraitCont(taxa, rate = 1, meanChange = 0, rootTrait = 0)
```

Arguments

`taxa`	A five-column matrix of taxonomic data, as output by `fossilRecord2fossilTaxa` after simulation with `simFossilRecord` (or via the deprecated function `simFossilTaxa`)
`rate`	rate of trait change; variance of evolutionary change distribution per speciation event
`meanChange`	Mean change per speciation event. Default is 0; change to simulate 'active' speciational trends, where the expected change at each speciational event is non-zero.
`rootTrait`	The trait value of the first taxon in the dataset; set to 0 by default.

Details

This function simulates continuous trait evolution where change occurs under a Brownian model, but only at events that create new distinct morphotaxa (i.e. species as recognized in the fossil record), either branching events or anagenesis (pseudospeciation). These are the types of morphological differentiation which can be simulated in the function `simFossilRecord`. This is sometimes referred to as cladogenetic or speciation trait evolution and is related to Puncuated Equilibrium theory. Anagenestic shifts aren't cladogenetic events per se (no branching!), so perhaps the best way to this of this function is it allows traits to change anytime `simFossilRecord` created a new 'morphotaxon' in a simulation.

Importantly, trait changes only occur at the base of 'new' species, thus allowing cladogenetic trait evolution to be asymmetrical at branching points: i.e. only one branch actually changes position in trait-space, as expected under a budding cladogenesis model. This distinction is important as converting the taxa matrix to a phylogeny and simulating the trait changes under a 'speciational' tree-transformation would assume that divergence occurred on both daughter lineages at each node. (This has been the standard approach for simulating cladogenetic trait change on trees).

Cryptic taxa generated with `prop.cryptic` in `simFossilRecord` will not differ at all in trait values. These species will all be identical.

See this link for additional details: http://nemagraptus.blogspot.com/2012/03/simulating-budding-cladogenetictrait.html

Value

Retuns a vector of trait values for each taxon, with value names being the taxa IDs (column 1 of the input) with a 't' pasted (as with rtree in the ape library).

Author(s)

David W. Bapst

See Also

`simFossilRecord`,

This function is similar to Brownian motion simulation functions such as `rTraitCont` in ape, `sim.char` in geiger and `fastBM` in phytools.

See also `unitLengthTree` in this package and `speciationalTree` in the package geiger. These are tree transformation functions; together with BM simulation functions, they would be expected to have a similar effect as this function (when cladogenesis is 'bifurcating' and not 'budding'; see above).

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,1000), plot=TRUE)
taxa<-fossilRecord2fossilTaxa(record)
trait <- cladogeneticTraitCont(taxa)
tree <- taxa2phylo(taxa)
```

```
plotTraitgram(trait,tree,conf.int=FALSE)

#with cryptic speciation
record<-simFossilRecord(p=0.1, q=0.1, prop.cryptic=0.5,
nruns=1, nTotalTaxa=c(30,1000), plot=TRUE)
taxa<-fossilRecord2fossilTaxa(record)
trait <- cladogeneticTraitCont(taxa)
tree <- taxa2phylo(taxa)
plotTraitgram(trait,tree,conf.int=FALSE)
```

communityEcology *Miscellaneous Functions for Community Ecology*

Description

This is just a small collection of miscellaneous functions that may be useful, primarily for community ecology analyses, particularly for paleoecological data. They are here mainly for pedagogical reasons (i.e. for students) as they don't appear to be available in other ecology-focused packages.

Usage

```
pairwiseSpearmanRho(x, dropAbsent = "bothAbsent", asDistance = FALSE,
  diag = NULL, upper = NULL, na.rm = FALSE)

HurlbertPIE(x, nAnalyze = Inf)
```

Arguments

`x`	The community abundance matrix. Must be a matrix with two dimensions for `pairwiseSpearmanRho`; if a vector is supplied to `HurlbertPIE`, then it is treated as if it was matrix with a single row and number of columns equal to its length. Taxonomic units are assumed to be the columns and sites (samples) are assumed to be the rows, for both functions.
`dropAbsent`	Should absent taxa be dropped? Must be one of either 'bothAbsent' (drop taxa absent in both sites for a given pairwise comparison),'eitherAbsent' (drop taxa absent in either site), or 'noDrop' (drop none of the taxa). The default 'bothAbsent' is recommended, see examples.
`asDistance`	Should the rho coefficients be rescaled on a scale similar to dissimilarity metrics, i.e. bounded 0 to 1, with 1 representing maximum dissimilarity (i.e. a Spearman rho correlation of -1)? (dissimilarity = (1 - rho) / 2)
`diag`	Should the diagonal of the output distance matrix be included?
`upper`	Should the upper triangle of the output distance matrix be included?
`na.rm`	Should taxa listed with NA values be dropped from a pair-wise site comparison? If FALSE, the returned value for that site pair will be NA if NAs are present.

`nAnalyze`	Allows users to select that PIE be calculated only on `nAnalyze` most abundant taxa in a site sample. `nAnalyze` must be a vector of length 1, consisting of whole-number value at least equal to 2. By default, `nAnalyze = Inf` so all taxa are accepted. Note that if there are less taxa in a sample than nAnalyze, the number present will be used.

Details

`pairwiseSpearmanRho` returns Spearman rho correlation coefficients based on the rank abundances of taxa (columns) within sites (rows) from the input matrix, by internally wrapping the function `cor.test`. It allows for various options that ultimatically allow for dropping taxa not shared between two sites (the default), as well as several other options. This allows the rho coefficient to behave like the Bray-Curtis distance, in that it is not affected by the number of taxa absent in both sites.

`pairwiseSpearmanRho` can also rescale the rho coefficients with (1-rho)/2 to provide a measure similar to a dissimilarity metric, bounded between 0 and 1. This function was written so several arguments would be in a similar format to the `vegan` library function `vegdist`. If used to obtain rho rescaled as a dissimilarity, the default output will be the lower triangle of a distance matrix object, just as is returned by default by `vegdist`. This behavior can be modified via the arguments for including the diagonal and upper triangle of the matrix. Otherwise, a full matrix is returned (by default) if the `asDistance` argument is not enabled.

`HurlbertPIE` provides the Probability of Interspecific Encounter metric for relative community abundance data, a commonly used metric for evenness of community abundance data based on derivations in Hurlbert (1971). An optional argument allows users to apply Hurlbert's PIE to only a subselection of the most abundant taxa.

Value

`pairwiseSpearmanRho` will return either a full matrix (the default) or (if `asDistance` is true, a distance matrix, with only the lower triangle shown (by default). See details.

`HurlbertPIE` returns a named vector of PIE values for the input data.

Author(s)

David W. Bapst

References

Hurlbert, S. H. 1971. The nonconcept of species diversity: a critique and alternative parameters. *Ecology* 52(4):577-586.

See Also

Example dataset: `kanto`

Examples

```
# let's load some example data:
# a classic dataset collected by Satoshi and Okido from the Kanto region

data(kanto)

rhoBothAbsent<-pairwiseSpearmanRho(kanto,dropAbsent="bothAbsent")

#other dropping options
rhoEitherAbsent<-pairwiseSpearmanRho(kanto,dropAbsent="eitherAbsent")
rhoNoDrop<-pairwiseSpearmanRho(kanto,dropAbsent="noDrop")

#compare
layout(1:3)
lim<-c(-1,1)
plot(rhoBothAbsent, rhoEitherAbsent, xlim=lim, ylim=lim)
abline(0,1)
plot(rhoBothAbsent, rhoNoDrop, xlim=lim, ylim=lim)
abline(0,1)
plot(rhoEitherAbsent, rhoNoDrop, xlim=lim, ylim=lim)
abline(0,1)
layout(1)

#using dropAbsent="eitherAbsent" reduces the number of taxa so much that
# the number of taxa present drops too low to be useful
#dropping none of the taxa restricts the rho measures to high coefficients
# due to the many shared 0s for absent taxa

#############

# Try the rho coefficients as a rescaled dissimilarity
rhoDist<-pairwiseSpearmanRho(kanto,asDistance=TRUE,dropAbsent="bothAbsent")

# What happens if we use these in typical distance matrix based analyses?

# Cluster analysis
clustRes<-hclust(rhoDist)
plot(clustRes)

# Principle Coordinates Analysis
pcoRes <- pcoa(rhoDist,correction="lingoes")
scores <- pcoRes$vectors
#plot the PCO
plot(scores,type="n")
text(labels=rownames(kanto),scores[,1],scores[,2],cex=0.5)

##################################

# measuring evenness with Hurlbert's PIE

kantoPIE<-HurlbertPIE(kanto)
```

```
#histogram
hist(kantoPIE)
#evenness of the kanto data is fairly high

#barplot
parX<-par(mar=c(7,5,3,3))
barplot(kantoPIE,las=3,cex.names=0.7,
ylab="Hurlbert's PIE",ylim=c(0.5,1),xpd=FALSE)
par(parX)

#and we can see that the Tower has extremely low unevenness
#...overly high abundance of ghosts?

#let's look at evenness of 5 most abundant taxa
kantoPIE_5<-HurlbertPIE(kanto,nAnalyze=5)

#barplot
parX<-par(mar=c(7,5,3,3))
barplot(kantoPIE_5,las=3,cex.names=0.7,
ylab="Hurlbert's PIE for 5 most abundant taxa",ylim=c(0.5,1),xpd=FALSE)
par(parX)
```

compareTimescaling *Comparing the Time-Scaling of Trees*

Description

These functions take two trees and calculate the changes in node ages (for compareNodeAges) for shared clades or terminal branch lengths leading to shared tip taxa (for compareTermBranches).

Usage

```
compareNodeAges(tree1, tree2, dropUnshared = FALSE)

compareTermBranches(tree1, tree2)
```

Arguments

`tree1`	A time-scaled phylogeny of class 'phylo'
`tree2`	A time-scaled phylogeny of class 'phylo'; for compareNodeAges, tree2 can also be an object of class 'multiPhylo' composed of multiple phylogenies. See below.
`dropUnshared`	If TRUE, nodes not shared across all input trees are dropped from the final output for compareNodeAge. This argument has no effect if tree2 is a single phylogeny (a 'phylo'-class object).

Details

For their most basic usage, these functions compare the time-scaling of two trees. Any taxa not-shared on both trees are dropped before analysis, based on tip labels.

As with many paleotree functions, calculations relating to time on trees are done with respect to any included $root.time elements. If these are not present, the latest tip is assumed to be at the present day (time=0).

compareNodeAges calculates the changes in the clade ages among those clades shared by the two trees, relative to the first tree in absolute time. For example, a shift of +5 means the clade originates 5 time-units later in absolute time on the second tree, while a shift of -5 means the clade originated 5 time-units prior on the second tree.

For compareNodeAges, if tree2 is actually a multiPhylo object composed of multiple phylogenies, the output will be a matrix, with each row representing a different tree and each column a different clade shared between at least some subset of the trees in tree2 and the tree in tree1. values in the matrix are the changes in clade ages between from tree1 (as baseline) to tree2, with NA values representing a clade that isn't contained in the tree represented by that row (but is contained in tree1 and at least one other tree in tree2). The matrix can be reduced to only those clades shared by all trees input via the argument dropUnshared. Note that this function distinguishes clades based on their shared taxa, and cannot infere that two clades might be identical if it were not for single taxon within the crown of one considered clade, despite that such a difference should probably have no effect on compare a node divergence date. Users should consider their dataset for such scenarios prior to application of compareNodeAges, perhaps by dropping all taxa not included in all other trees to be considered (this is NOT done by this function).

compareTermBranches calculates the changes in the terminal branch lengths attached to tip taxa shared by the two trees, relative to the first tree. Thus, a shift of +5 means that this particular terminal taxon is connected to a terminal branch which is five time-units longer.

Value

compareTermBranches returns a vector of temporal shifts for terminal branches with the shared tip names as labels.

compareNodeAges, if both tree1 and tree2 are single trees, outputs a vector of temporal shifts for nodes on tree2 with respect to tree1. If tree2 is multiple trees, then a matrix is output, with each row representing each tree in tree2 (and carrying the name of each tree, if any is given). The values are temporal shifts for each tree in tree2 with respect to tree1. For either case, the column names or element names (for a vector) are the sorted taxon names of the particular clade, the dates of which are given in that column. See above for more details. These names can be very long when large trees are considered.

See Also

`dateNodes`, `taxa2phylo`, `phyloDiv`

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#get the true tree
tree1 <- taxa2phylo(taxa)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot=TRUE)
#Now let's try timePaleoPhy using the continuous range data
tree2 <- timePaleoPhy(cladogram,rangesCont,type="basic")
#let's look at the distribution of node shifts
hist(compareNodeAges(tree1,tree2))
#let's look at the distribution of terminal branch lengths
hist(compareTermBranches(tree1,tree2))

#testing ability to compare multiple trees with compareNodeAges
trees <- cal3TimePaleoPhy(cladogram,rangesCont,brRate=0.1,extRate=0.1,
    sampRate=0.1,ntrees=10)
nodeComparison <- compareNodeAges(tree1,trees)
#plot it as boxplots for each node
boxplot(nodeComparison,names=NULL);abline(h=0)
#plot mean shift in node dates
abline(h=mean(apply(nodeComparison,2,mean,na.rm=TRUE)),lty=2)

#just shifting a tree back in time
set.seed(444)
tree1 <- rtree(10)
tree2 <- tree1
tree1$root.time <- 10
compareNodeAges(tree1,tree2)
compareTermBranches(tree1,tree2)
```

constrainParPaleo *Constrain Parameters for a Model Function from paleotree*

Description

This function constrains a model to make submodels with fewer parameters, using a structure and syntax taken from the function `constrain` in Rich Fitzjohn's package `diversitree`.

Usage

```
constrainParPaleo(f, ..., formulae = NULL, names = parnames(f),
  extra = NULL)
```

Arguments

`f`	A function to constrain. This function must be of S3 class 'paleotreeFunc' and have all necessary attributes expected of that class, which include parameter

	names and upper and lower bounds. As I have deliberately not exported the function which creates this class, it should be impossible for regular users to obtain such objects easily without using one of the 'make' functions, which automatically output a function of the appropriate class and attributes.
`...`	Formulae indicating how the function should be constrained. See details and examples for lengthy discussion.
`formulae`	Optional list of constraints, possibly in addition to those in `...`
`names`	Optional Character vector of names, the same length as the number of parameters in `x`. Use this only if `parnames` does not return a vector for your function. Generally this should not be used. DWB: This argument is kept for purposes of keeping the function as close to the parent as possible but, in general, should not be used because the input function must have all attributes expected of class 'paleotreeFunc', including parameter names.
`extra`	Optional vector of additional names that might appear on the RHS of constraints but do not represent names in the function's `argnames`. This can be used to set up dummy variables (example coming later).

Details

This function is based on (but does not depend on) the function `constrain` from the package `diversitree`. Users should refer to this parent function for more detailed discussion of model constraint usage and detailed examples.

The parent function was forked to add functionality necessary for dealing with the high parameter count models typical to some paleontological analyses, particularly the inverse survivorship method. This necessitated that the new function be entirely separate from its parent. Names of functions involved (both exported and not) have been altered to avoid overlap in the package namespaces. Going forward, the paleotree package maintainer (Bapst) will try to be vigilant with respect to changes in `constrain` in the original package, `diversitree`.

Useful information from the diversitree manual (11/01/13):

"If `f` is a function that takes a vector `x` as its first argument, this function returns a new function that takes a shorter vector `x` with some elements constrained in some way; parameters can be fixed to particular values, constrained to be the same as other parameters, or arbitrary expressions of free parameters."

In general, formulae should be of the structure:

LHS ~ RHS

...where the LHS is the 'Parameter We Want to Constrain' and the RHS is whatever we are constraining the LHS to, usually another parameter. LHS and RHS are the 'left-hand side' and 'right-hand side' respectively (which I personally find obscure).

Like the original `constrain` function this function is based on, this function cannot remove constraints previously placed on a model object and there may be cases in which the constrained function may not make sense, leading to an error. The original function will sometimes issue nonsensical functions with an incorrect number/names of parameters if the parameters to be constrained are given in the wrong order in formulae.

Differences from diversitree's constrain Function:

This forked paleotree version of constrain has two additional features, both introduced to aid in constraining models with a high number of repetitive parameters. (I didn't invent these models, don't shoot the messenger.)

First, it allows nuanced control over the constraining of many parameters simultaneously, using the 'all' and 'match' descriptors. This system depends on parameters being named as such: name.group1.group2.group3 and so on. Each 'group' is reference to a system of groups, perhaps referring to a time interval, region, morphology, taxonomic group or some other discrete characterization among the data (almost all functions envisioned for paleotree are for per-taxon diversification models). The number of group systems is arbitrary, and may be from zero to a very large number; it depends on the 'make' function used and the arguments selected by the user. For example, the parameter 'x.1' would be for the parameter 'x' in the first group of the first group system (generally a time interval for most paleotree functions). For a more complicated exampled, with the parameter 'x.1.3.1', the third group for the second group system (perhaps this taxonomic data point has a morphological feature not seen in some other taxa) and group 1 of the third group system (maybe biogeographic region 1? the possibilities are endless depending on user choices).

The 'all' option work like so: if 'x.all~x.1' is given as a formulae, then all x parameters will be constrained to equal x.1. For example, if there is x.1, x.2, x.3 and x.4 parameters for a model, 'x.all~x.1' would be equivalent to individually giving the formulae 'x.2~x.1', 'x.3~x.1' and 'x.4~x.1'. This means that if there are many parameters of a particular type (say, 50 'x' parameters) it is easy to constrain all with a short expression. It is not necessary that the The 'all' can be used anywhere in the name of the parameter in a formulae, including to make all parameters for a given 'group' the same. Furthermore, the LHS and RHS don't need to be same parameter group, and both can contain 'all' statements, even *multiple* 'all' statements. Consider these examples, all of which are legal:

x.all ~ y.1 Constrains all values of the parameter x for every group to be equal to the single value for the parameter y for group 1 (note that there's only a single set of groups).

all.1 ~ x.1 Constrains all parameters for the first group to equal each other, here arbitrary named x.1. For example, if there is parameters named x.1, y.1 and z.1, all will be constrained to be equal to a single parameter value.

x.all.all ~ y.2.3 Constrains all values for x in any and all groups to equal the single value for y for group 2 (of system 1) and group 3 (of system 2).

x.all ~ y.all Constrains all values of x for every group and y for every group to be equal to a *single* value, which by default will be reported as y.1

The 'match' term is similar, allowing parameter values from the same group to be quickly matched and made equivalent. These 'match' terms must have a matching (hah!) term both in the corresponding LHS and RHS of the formula. For example, consider 'x.match~y.match' where there are six parameters: x.1, x.2, x.3, y.1, y.2 and y.3. This will effectively constrain x.1~y.1, x.2~y.2 and x.3~y.3. This is efficient for cases where we have some parameters that we often treat as equal. For example, in paleontology, we sometimes make a simplifying assumption that birth and death rates are equal in multiple time intervals. Some additional legal examples are:

x.match.1 ~ y.match.1 This will constrain only parameters of x and y to to equal each other if they both belong to the same group for the first group system AND belong to group 1 of the first group.

all.match. ~ x.match This will constrain all named parameters in each group to equal each other; for example, if there are parameters x.1, y.1, z.1, x.2, y.2 and z.2, this will constrain them such that y.1~x.1, z.1~x.1, y.2~x.2 and z.2~x.2, leaving x.1 and x.2 as the only parameters effectively.

There are two less fortunate qualities to the introduction of the above terminology.

Unfortunately, this package author apologizes that his programming skills are not good enough to allow more complex sets of constraints, as would be typical with the parent function, when 'all' or 'match' terms are included. For example, it would not be legal to attempt to constraint 'y.all ~ x.1 / 2', where the user presumably is attempting to constrain all y values to equal the x parameter to equal half of the x parameter for group 1. This will not be parsed as such and should return an error. However, there are workarounds, but they require using constrainParPaleo more than once. For the above example, a user could first use 'y.all ~ y.1' constraining all y values to be equal. Then a user could constrain with the formula 'y.1 ~ x.1 / 2' which would then constrain y.1 (and all the y values constrained to equal it) to be equal to the desired fraction.

Furthermore, this function expects that parameter names don't already have period-separated terms that are identical to 'all' or 'match'. No function in paleotree should produce such natively. If such were to occur, perhaps by specially replacing parameter names, constrainParPaleo would confuse these terms for the specialty terms described here.

Secondly, this altered version of constrain handles the parameter bounds included as attributes in functions output by the various 'make' functions. This means that if 'x.1 ~ y.1', constrainParPaleo will test if the bounds on x.1 and y.1 are the same. If the bounds are not the same, constrainParPaleo will return an error. This is important, as some models in paleotree may make a parameter a rate (bounded zero to some value greater than one) or a probability (bounded zero to one), depending on user arguments. Users may not realize these differences and, in many cases, constraining a rate to equal a probability is nonsense (absolute poppycock!). If a user really wishes to constrain two parameters with different bounds to be equal (I have no idea why anyone would want to do this), they can use the parameter bound replacement functions described in `modelMethods` to set the parameter bounds as equal. Finally, once parameters with the same bounds are constrained, the output has updated bounds that reflect the new set of parameters for the new constrained function.

Value

Modified from the diversitree manual: This function returns a constrained function that can be passed through to the optimization functions of a user's choice, such as `optim`, `find.mle` in diversitree or `mcmc`. It will behave like any other function. However, it has a modified `class` attribute so that some methods will dispatch differently: `parnames`, for example, will return the names of the parameters of the constrained function and `parInit` will return the initial values for those same constrained set of parameters. All arguments in addition to `x` will be passed through to the original function `f`.

Additional useful information from the diversitree manual (11/01/13):

For help in designing constrained models, the returned function has an additional argument `pars.only`, when this is `TRUE` the function will return a named vector of arguments rather than evaluate the function (see Examples).

Author(s)

This function (and even this help file!) was originally written by Rich Fitzjohn for his library `diversitree`, and subsequently rewritten and modified by David Bapst.

References

FitzJohn, R. G. 2012. Diversitree: comparative phylogenetic analyses of diversification in R. *Methods in Ecology and Evolution* 3(6):1084-1092.

See Also

As noted above, this function is based on (but does not depend on) the function `constrain` from the library `diversitree`.

Examples

```
#simulation example with make_durationFreqCont, with three random groups
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r=0.5)
grp1 <- matrix(sample(1:3,nrow(taxa),replace=TRUE),,1)   #groupings matrix
likFun <- make_durationFreqCont(rangesCont,groups=grp1)

# can constrain both extinction rates to be equal
constrainFun <- constrainParPaleo(likFun,q.2~q.1)

#see the change in parameter names and bounds
parnames(likFun)
parnames(constrainFun)
parbounds(likFun)
parbounds(constrainFun)

# some more ways to constrain stuff!

#constrain all extinction rates to be equal
constrainFun <- constrainParPaleo(likFun,q.all~q.1)
parnames(constrainFun)

#constrain all rates for everything to be a single parameter
constrainFun <- constrainParPaleo(likFun,r.all~q.all)
parnames(constrainFun)

#constrain all extinction rates to be equal & all sampling to be equal
constrainFun <- constrainParPaleo(likFun,q.all~q.1,r.all~r.1)
parnames(constrainFun)

#similarly, can use match.all to make all matching parameters equal each other
constrainFun <- constrainParPaleo(likFun,match.all~match.all)
parnames(constrainFun)

#Constrain rates in same group to be equal
constrainFun <- constrainParPaleo(likFun,r.match~q.match)
parnames(constrainFun)
```

`createMrBayesConstraints` *Transform a Topology into a Set of Constraint Commands for MrBayes*

Description

Takes a phylogeny in the form of an object of class `phylo` and outputs a set of topological constraints for *MrBayes* as a set of character strings, either printed in the R console or in a named text file, which can be used as commands in the *MrBayes* block of a NEXUS file for use with (you guessed it!) *MrBayes*.

Usage

```
createMrBayesConstraints(tree, partial = TRUE, file = NULL)
```

Arguments

`tree`	An object of class `phylo`.
`partial`	If `TRUE` (the default), then constraints will be defined as partial constraints with respect to the rest of the taxa in the input `tree`. If `FALSE`, constraints will be defined as hard clade membership constraints (i.e. no additional taxa will be allowed to belong to the nodes present in the observed tree. Depending on your analysis, `partial = TRUE` may require additionally defining an outgroup.
`file`	Filename (possibly with path) as a character string to a file which will be overwritten with the output constraint lines. If not null, not constraint lines are output to the console.

Details

`partial = TRUE` may be useful if the reason for using `createMrBayesConstraints` is to constrain a topology containing some of the taxa in an analysis, while allowing other taxa to freely vary. For example, Slater (2013) constrained an analysis so extant taxon relationships were held constant, using a molecular-based topology, while allowing fossil taxa to freely vary relative to their morphological character data.

Value

If argument `file` is `NULL`, then the constrain commands are ouput as a series of character strings.

Author(s)

David W. Bapst, with some inspiration from Graham Slater.

References

Slater, G. J. 2013. Phylogenetic evidence for a shift in the mode of mammalian body size evolution at the Cretaceous-Palaeogene boundary. *Methods in Ecology and Evolution* 4(8):734-744.

Examples

```
set.seed(444)
tree<-rtree(10)
createMrBayesConstraints(tree)
createMrBayesConstraints(tree,partial=FALSE)

## Not run:

createMrBayesConstraints(tree,file="topoConstraints.txt")


## End(Not run)
```

`dateNodes` *Absolute Dates for Nodes of a Time-Scaled Phylogeny*

Description

This function returns the ages of nodes (both internal and terminal tips) for a given phylogeny of class 'phylo'. Specialized for use with time-scaled trees from paleotree, see Details.

Usage

```
dateNodes(tree, rootAge = tree$root.time, labelDates = FALSE,
  tolerance = 0.001)
```

Arguments

`tree`	A phylogeny object of class 'phylo'. Must have edge.lengths!
`rootAge`	The root age of the tree, assumed by default to be equal to the element tree$root.time, which is a standard element for trees time-scaled by the paleotree package. If not given by the user and if the $root.time element does not exist, then the maximum depth of the tree will be taken as the root age, which implicitly assumes the latest most terminal tip is an extant taxon at the modern day (time = 0). If rootAge is so defined that some nodes may occur later than time = 0, this function may return negative dates.
`labelDates`	If FALSE (the default), the dates returned are labeled with the tip/node numbers as in `tree$edge`. If TRUE, they are labeled with the tip labels of every descendant tip, which for terminal tips means a single taxon label, and for internal tips a label that might be very long, composed of multiple tip labels pasted together. Thus, by default, this argument is FALSE.
`tolerance`	The tolerance within which a node date has to be removed from time = 0 (i.e. the modern) to issue a warning that there are 'negative' node dates.

Details

This function is specialized for phylo objects time-scaled or simulated with functions from paleotree, and thus have a $root.time element. This function will still work without such, but users should see the details for the `rootAge` argument.

Value

Returns a vector of length `Ntip(tree) + Nnode(tree)` which contains the dates for all terminal tip nodes and internal nodes for the tree, in that order, as numbered in the `tree$edge` matrix. These dates are always on a descending scale (i.e. time before present); see rootAge for how the present time is determined. If rootAge is so defined that some nodes may occur later than time = 0 units before present, this function may (confusingly) return negative dates and a warning message will be issued.

Author(s)

David W. Bapst, based on a function originally written by Graeme Lloyd.

See Also

`compareTimescaling`

Examples

```
#let's simulate some example data
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#get the true time-sclaed tree
tree1 <- taxa2phylo(taxa)

#now let's try dateNodes
dateNodes(tree1)

#let's ignore $root.time
dateNodes(tree1,rootAge=NULL)

#with the lengthy tip-label based labels
   #some of these will be hideously long
dateNodes(tree1,labelDates=TRUE)
```

`degradeTree` *Randomly Collapse a Portion of Nodes on a Phylogeny*

Description

`degradeTree` removes a proportion of the total nodes in a tree, chosen randomly, collapsing the nodes to produce a less-resolved tree. The related function `collapseNodes` given a tree and a vector of nodes to collapse, removes those nodes from a tree, creating a polytomy.

Usage

```
degradeTree(tree, prop_collapse = NULL, nCollapse = NULL, node.depth = NA,
  leave.zlb = FALSE)

collapseNodes(tree, nodeID, collapseType, leave.zlb = FALSE)
```

Arguments

`tree`	A phylogeny of class 'phylo'
`prop_collapse`	Proportion of nodes to collapse
`nCollapse`	Number of nodes to collapse, can be supplied as an alternative to prop_collapse
`node.depth`	A number between 0 to 1, which conditions the depth of nodes removed. If NA, no conditioning (this is the default).
`leave.zlb`	If FALSE, the default option, the original branch length distribution is destroyed and branches set to zero by this function will return polytomies. If TRUE, then the original edge lengths are kept for unmodified edges, and modified edges are changed to zero length, and are not collapsed into polytomies. The removed branch length is not shifted to other edges.
`nodeID`	The node ID number(s) to be collapsed into a polytomy, as identified in the `$edge` matrix of the'phylo' object. Must be a vector of one or more ID numbers.
`collapseType`	Whether to collapse the edge leading the listed node (if "forward"), or to collapse the child edges leading away from the node (if "backward"). Collapsing a node 'into' a polytomy conceptually could be either and users should heed this option carefully. A third option, if "collapseType=clade" is to collapse the entire clade that is descended from a node (i.e. forward).

Details

In the function `degradeTree`, the nodes are removed at random using the basic R function `sample`. `degradeTree` can be conditioned to remove nodes of a particular depth with greater probability/frequency by setting node.depth to a value between zero (favoring the removal of deep nodes close to the root) or one (shallow nodes far from the root). Depth is evaluated based on the number of descendant tips. If node.depth is not NA, the relative proportion of descendants from each node is calculated, summed to 1 and the node.depth value subtracted from this proportion. These values are then squared, normalized again to equal to 1 and then used as the probabilities for sampling nodes for removal.

By default, branch lengths are removed from the input tree prior to degradation and entirely absent from the output tree. This is changed if argument leave.zlb is TRUE.

Value

Returns the modified tree as an object of class phylo, with no edge lengths by default.

Author(s)

David W. Bapst

See Also

di2multi,timeLadderTree

Examples

```
set.seed(444)
tree <- rtree(100)
tree1 <- degradeTree(tree,prop_collapse=0.5)
tree3 <- degradeTree(tree,nCollapse=50)

#let's compare the input and output
layout(matrix(1:2,,2))
plot(tree,show.tip.label=FALSE,use.edge.length=FALSE)
plot(tree1,show.tip.label=FALSE,use.edge.length=FALSE)

#now with collapseNodes
tree <- rtree(10)
#collapse nodes backwards
   #let's collapse lucky node number 13!
tree1 <- collapseNodes(nodeID=13,tree=tree,collapseType="backward")
#collapse nodes forwards
tree2 <- collapseNodes(nodeID=13,tree=tree,collapseType="forward")
#collapse entire clade
tree3 <- collapseNodes(nodeID=13,tree=tree,collapseType="clade")

#let's compare
layout(1:4)
plot(tree,use.edge.length=FALSE,main="original")
plot(tree1,use.edge.length=FALSE,main="backward collapse")
plot(tree2,use.edge.length=FALSE,main="forward collapse")
plot(tree3,use.edge.length=FALSE,main="entire clade")

layout(1)
```

---

depthRainbow *Paint Tree Branch Depth by Color*

Description

Paints the edges of a phylogeny with colors relative to their depth.

Usage

```
depthRainbow(tree)
```

Arguments

tree	A phylo object

Details

The only purpose of this function is to make an aesthetically-pleasing graphic of one's tree, where branches are color-coded with a rainbow palette, relative to their depth. Depth is defined relative to the number of branching nodes between the basal node of a branch and the root, not the absolute distance (i.e. branch length) to the root or the distance from the tips.

Value

No value returned, just plots a colorful phylogeny.

Examples

```
set.seed(444)
tree <- rtree(500)
depthRainbow(tree)
```

`divCurveFossilRecordSim` *Diversity-Curve Plotting for Simulations of Diversification and Sampling In the Fossil Record*

Description

An extremely simple plotting function, which plots the original taxonomic diversity versus the sampled taxonomic diversity, for use with output from the function `simFossilRecord`. If sampling processes were not included in the model, then it plots simply the single diversity curve.

Usage

```
divCurveFossilRecordSim(fossilRecord, merge.cryptic = TRUE,
  plotLegend = TRUE, legendPosition = "topleft", curveColors = c("black",
  "red"), curveLineTypes = c(1, 2))
```

Arguments

`fossilRecord`	A list object output by `simFossilRecord`, often composed of multiple elements, each of which is data for 'one taxon', with the first element being a distinctive six-element vector composed of numbers, corresponding to the six variable tables by `fossilRecord2fossilTaxa` after simulating with `simFossilRecord` (originally produced by deprecated function `simFossilTaxa`).
`merge.cryptic`	If `TRUE`, cryptic taxon-units (i.e. those in the same cryptic complex) will be merged into single taxa for the sake of being counted in the diversity curves presented by this function.
`plotLegend`	A logical. Should a legend be plotted? Only applies if sampling processes were modeled.

legendPosition	Where should the legend be plotted? See help for legend for details. Only applies if sampling processes were modeled.
curveColors	A vector of length two indicating what colors the original and sampled diversity curves should be displayed in. Only applies if sampling processes were modeled.
curveLineTypes	A vector of length two indicating what colors the original and sampled diversity curves should be displayed in. Only applies if sampling processes were modeled.

Details

This function is essentially a wrapper for `paleotree` function `multiDiv`.

Value

This function returns nothing: it just creates a plot.

Author(s)

David W. Bapst

See Also

simFossilRecord

Examples

```
set.seed(44)
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
nTotalTaxa=c(20,30) ,nExtant=0, plot=FALSE)

# now let's plot it
divCurveFossilRecordSim(record)
```

`DiversityCurves` *Diversity Curves*

Description

Functions to plot diversity curves based on taxic range data, in both discrete and continuous time, and for phylogenies.

Usage

```
taxicDivCont(timeData, int.length = 1, int.times = NULL, plot = TRUE,
  plotLogRich = FALSE, timelims = NULL, drop.cryptic = FALSE)

taxicDivDisc(timeList, int.times = NULL, drop.singletons = FALSE,
  plot = TRUE, plotLogRich = FALSE, timelims = NULL,
  extant.adjust = 0.001, split.int = TRUE)

phyloDiv(tree, int.length = 0.1, int.times = NULL, plot = TRUE,
  plotLogRich = FALSE, drop.ZLB = TRUE, timelims = NULL)
```

Arguments

`timeData`	Two-column matrix giving the per-taxon first and last appearances in absolute time. The simulated data tables output by `fossilRecord2fossilTaxa` following simulation with `simFossilRecord` can also be supplied to `taxicDivCont`.
`int.length`	The length of intervals used to make the diversity curve. Ignored if int.times is given.
`int.times`	An optional two-column matrix of the interval start and end times for calculating the diversity curve. If NULL, calculated internally. If given, the argument split.int and int.length are ignored.
`plot`	If TRUE, a diversity curve generated from the data is plotted.
`plotLogRich`	If TRUE, taxic diversity is plotted on log scale.
`timelims`	Limits for the x (time) axis for diversity curve plots. Only affects plotting. Given as either NULL (the default) or as a vector of length two as for 'xlim' in the basic R function plot. Time axes will be plotted *exactly* to these values.
`drop.cryptic`	If TRUE, cryptic taxa are merged to form one taxon for estimating taxon curves. Only works for objects from `simFossilRecord` via `fossilRecord2fossilTaxa`.
`timeList`	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. See details.
`drop.singletons`	If TRUE, taxa confined to a single interval will be dropped prior to the diversity curve calculation. This is sometimes done if single intervals have overly high diversities due to the 'monograph' effect where more named taxa are known in certain intervals largely due to taxonomic expert effort and not real changes in historical biotic diversity.
`extant.adjust`	Amount of time to be added to extend start time for (0,0) bins for extant taxa, so that the that 'time interval' doesn't appear to have an infinitely small width.
`split.int`	For discrete time data, should calculated/input intervals be split at discrete time interval boundaries? If FALSE, can create apparent artifacts in calculating the diversity curve. See below.
`tree`	A time-scaled phylogeny of class phylo.
`drop.ZLB`	If true, zero-length terminal branches are dropped from the input tree for phylogenetic datasets, before calculating standing diversity.

Details

First, some background. Diversity curves are plots of species/taxon/lineage richness over time for a particular group of organisms. For paleontological studies, these are generally based on per-taxon range data while more recently in evolutionary biology, molecular phylogenies have been used to calculate lineage-through-time plots (LTTs). Neither of these approaches are without their particular weaknesses; reconstructing the true history of biodiversity is a difficult task no matter what data is available.

The diversity curves produced by these functions will always measure diversity within binned time intervals (and plot them as rectangular bins). For continuous-time data or phylogenies, one could decrease the int.length used to get what is essentially an 'instantaneous' estimate of diversity. This is warned against, however, as most historical diversity data will have some time-averaging or uncertain temporal resolution and thus is probably not finely-resolved enough to calculate instantaneous estimates of diversity.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

As diversity is counted within binned intervals, the true standing diversity may be somewhat lower than the measured (observed) quantity, particularly if intervals are longer than the mean duration of taxa is used. This will be an issue with all diversity curve functions, but particularly the discrete-time variant. For diversity data in particularly large discrete time intervals, plotting this data in smaller bins which do not line up completely with the original intervals will create a 'spiky' diversity curve, as these smaller intersecting bins will have a large number of taxa which may have been present in either of the neighboring intervals. This will give these small bins an apparently high estimated standing diversity. This artifact is avoided with the default setting split.int=TRUE, which will split any input or calculated intervals so that they start and end at the boundaries of the discrete-time range bins.

The timeList object should be a list composed of two matrices, the first matrix giving by-interval start and end times (in absolute time), the second matrix giving the by-taxon first and last appearances in the intervals defined in the first matrix, numbered as the rows. Absolute time should be decreasing, while the intervals should be numbered so that the number increases with time. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. See the documentation for the time-scaling function `bin_timePaleoPhy` and the simulation function `binTimeData` for more information on formatting.

Unlike some paleotree functions, such as perCapitaRates, the intervals can be overlapping or of unequal length. The diversity curve functions deal with such issues by assuming taxa occur from the base of the interval they are first found in until the end of the last interval they are occur in. Taxa in wide-ranging intervals that contain many others will be treated as occurring in all nested intervals.

phyloDiv will resolve polytomies to be dichotomous nodes separated by zero-length branches prior to calculating the diversity curve. There is no option to alter this behavior, but it should not affect the use of the function because the addition of the zero-length branches should produce an identical diversity history as a polytomy. phyloDiv will also drop zero-length terminal branches, as with the function dropZLB. This the default behavior for the function but can be turned off by setting the argument drop.zlb to FALSE.

Value

These functions will invisibly return a three-column matrix, where the first two columns are interval start and end times and the third column is the number of taxa/lineages counted in that interval.

Author(s)

David W. Bapst

See Also

`multiDiv`, `timeSliceTree`, `binTimeData`

There are several different functions for traditional LTT plots (phylogenetic diversity curves), such as the function ,`ltt.plot` in the package ape, the function `ltt` in the package phytools, the function `plotLtt` in the package laser and the function `LTT.average.root` in the package TreeSim.

Examples

```
#taxicDivDisc with the retiolinae dataset
data(retiolitinae)
taxicDivDisc(retioRanges)

#simulation examples
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#let's see what the 'true' diversity curve looks like in this case
#plot the FADs and LADs with taxicDivCont()
taxicDivCont(taxa)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.5)
#plot the diversity curve based on the sampled ranges
layout(1:2)
taxicDivCont(rangesCont)
#Now let's use binTimeData to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length=1)
#plot with taxicDivDisc
taxicDivDisc(rangesDisc)
#compare to the continuous time diversity curve

layout(1)
#Now let's make a tree using taxa2phylo
tree <- taxa2phylo(taxa,obs_time=rangesCont[,2])
phyloDiv(tree)

#a simple example with phyloDiv
  #using a tree from rtree in ape
set.seed(444)
tree <- rtree(100)
phyloDiv(tree)
```

```
#a neat example of using phyDiv with timeSliceTree
 #to simulate doing molecular-phylogeny studies
 #of diversification...in the past
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
taxicDivCont(taxa)
#that's the whole diversity curve
#with timeSliceTree we could look at the lineage accumulation curve
 #we'd get of species sampled at a point in time
tree <- taxa2phylo(taxa)
#use timeSliceTree to make tree of relationships up until time=950
tree950 <- timeSliceTree(tree,sliceTime=950,plot=TRUE,drop.extinct=FALSE)
#use drop.extinct=T to only get the tree of lineages extant at time=950
tree950 <- timeSliceTree(tree,sliceTime=950,plot=TRUE,drop.extinct=TRUE)
#now its an ultrametric tree with many fewer tips...
#lets plot the lineage accumulation plot on a log scale
phyloDiv(tree950,plotLogRich=TRUE)

#an example of a 'spiky' diversity curve and why split.int is a good thing
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
taxaDiv <- taxicDivCont(taxa)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5)
rangesDisc <- binTimeData(rangesCont,int.length=10)
#now let's plot with taxicDivDisc() but with the intervals from taxaDiv
 #by default, split.int=TRUE
taxicDivDisc(rangesDisc,int.times=taxaDiv[,1:2],split.int=TRUE)
#look pretty
#now let's turn off split.int
taxicDivDisc(rangesDisc,int.times=taxaDiv[,1:2],split.int=FALSE)
#looks 'spiky'!
```

durationFreq *Models of Sampling and Extinction for Taxonomic Duration Datasets*

Description

These functions construct likelihood models of the observed frequency of taxon durations, given either in discrete (`make_durationFreqDisc`) or continuous time (`make_durationFreqCont`). These models can then be constrained using functions available in this package and/or analyzed with commonly used optimizing functions.

Usage

```
make_durationFreqCont(timeData, groups = NULL, drop.extant = TRUE,
  threshold = 0.01, tol = 1e-04)

make_durationFreqDisc(timeList, groups = NULL, drop.extant = TRUE)
```

Arguments

`timeData`	Two-column matrix of per-taxon first and last occurrence given in continuous time, relative to the modern (i.e. older dates are also the 'larger' dates). Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs the supplied matrix) are automatically dropped from the matrix and from groups simultaneously.
`groups`	Either NULL (the default) or matrix with the number of rows equal to the number of taxa and the number of columns equal to the number of 'systems' of categories for taxa. Taxonomic membership in different groups is indicated by numeric values. For example, a dataset could have a 'groups' matrix composed of a column representing thin and thick shelled taxa, coded 1 and 2 respectively, while the second column indicates whether taxa live in coastal, shelfal or deep marine settings, coded 1-3 respectively. Different combinations of groups will be treated as having independent sampling and extinction parameters in the default analysis, for example, thinly-shelled deep marine species will have separate parameters from thinly-shelled coastal species. Grouping systems could also represent temporal heterogeneity, for example, categorizing Paleozoic versus Mesozoic taxa. If groups are NULL (the default), all taxa are assumed to be of the same group with the same parameters. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in timeData or timeList) are automatically dropped from groupings and the time dataset (either timeData or timeList) and from groups simultaneously.
`drop.extant`	Drops all extant taxa from a dataset, w
`threshold`	The smallest allowable duration (i.e. the measured difference in the first and last occurrence dates for a given taxon). Durations below this size will be treated as "one-hit" sampling events.
`tol`	Tolerance level for determining whether a taxon from a continuous-time analysis is extant or not. Taxa which occur at a date less than tol are treated as occurring at the modern day (i.e. being functionally identical as occurring at 0 time).
`timeList`	A 2 column matrix with the first and last occurrences of taxa given in relative time intervals (i.e. ordered from first to last). If a list of length two is given for timeData, such as would be expected if the output of binTimeData was directly input, the second element is used. See details. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in the second matrix) are automatically dropped from the timeList and from groups simultaneously. Living taxa observed in the modern day are expected to be listed as last observed in a special interval (0,0), i.e. begins and ends at 0 time. This interval is always automatically removed prior to the calculation intermediary data for fitting likelihood functions.

Details

These functions effectively replace two older functions in paleotree, now removed, `getSampRateCont` and `getSampProbDisc`. The functions here do not offer the floating time interval options of their older siblings, but do allow for greater flexibility in defining constrains on parameter values. Differences in time intervals, or any other conceivable discrete differences in parameters, can be modeled using the generic `groups` argument in these functions.

These functions use likelihood functions presented by Foote (1997). These analyses are ideally applied to data from single stratigraphic section but potentially are applicable to regional or global datasets, although the behavior of those datasets is less well understood.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero and older dates are 'larger'. On the contrary, relative time is in intervals with non-zero integers that increase sequentially beginning with 1, from earliest to oldest.

For `make_durationFreqDisc`, the intervals in timeList should be non-overlapping sequential intervals of roughly equal length. These should be in relative time as described above, so the earliest interval should be 1 and the numbering should increase as the intervals go up with age. If both previous statements are true, then differences in interval numbers will represent the same rough difference in the absolute timing of those intervals. For example, a dataset where all taxa are listed from a set of sequential intervals of similar length, such as North American Mammal assemblage zones, microfossil faunal zones or graptolite biozones can be given as long as they are correctly numbered in sequential order in the input. As a counter example, a dataset which includes taxa resolved only to intervals as wide as the whole Jurassic and taxa resolved to biozones within the Jurassic should not be included in the same input. Drop taxa from less poorly resolved intervals from such datasets if you want to apply this function, as long as this retains a large enough sample of taxa listed from the sequential set of intervals.

Please check that the optimizer function you select actually converges. The likelihood surface can be very flat in some cases, particularly for small datasets (<100 taxa). If the optimizer does not converge, consider increasing iterations or changing the starting values.

Value

A function of class "paleotreeFunc", which takes a vector equal to the number of parameters and returns the *negative* log likelihood (for use with optim and similar optimizing functions, which attempt to minimize support values). See the functions listed at `modelMethods` for manipulating and examining such functions and `constrainParPaleo` for constraining parameters.

Parameters in the output functions are named 'q' for the instantaneous per-capita extinction rate, 'r' for the instantaneous per-capita sampling rate and 'R' for the per-interval taxonomic sampling probability. Groupings follow the parameter names, separated by periods; by default, the parameters will be placed in at least group 1 (of a grouping with a single group), such that make_durationFreqCont by default creates a function with parameters named 'q.1' and 'r.1', while make_durationFreqDisc creates a function with parameters named 'q.1' and 'R.1'.

Note that the 'q' parameters estimated by `make_durationFreqDisc` is scaled to per lineage intervals and not to per lineage time-units. If intervals are the same length, this can be easily corrected by multiplying 1 by the interval length. It is unclear how to treat uneven intervals and I urge workers to consider multiple strategies.

For translating these sampling probabilities and sampling rates, see `SamplingConv`.

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

See Also

See `freqRat`, `sRate2sProb`, `qsRate2Comp` `sProb2sRate` and `qsProb2Comp`.

Examples

```
#let's simulate some taxon ranges from an imperfectly sampled fossil record
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r=0.5)
#bin the ranges into discrete time intervals
rangesDisc <- binTimeData(rangesCont,int.length=1)
#note that we made interval lengths = 1:
  # thus q (per int) = q (per time) for make_durationFreqDisc

## Not run:
#old ways of doing it (defunct as of paleotree version 2.6)
getSampRateCont(rangesCont)
getSampProbDisc(rangesDisc)

## End(Not run)

#new ways of doing it
    # we can constrain our functions
    # we can use parInit, parLower and parUpper to control parameter bounds

#as opposed to getSampRateCont, we can do:
likFun<-make_durationFreqCont(rangesCont)
optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
      method="L-BFGS-B",control=list(maxit=1000000))

#as opposed to getSampProbDisc, we can do:
likFun<-make_durationFreqDisc(rangesDisc)
optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
     method="L-BFGS-B",control=list(maxit=1000000))

#these give the same answers (as we'd expect them to!)

#with newer functions we can constrain our functions easily
```

```
    # what if we knew the extinction rate = 0.1 a priori?
likFun<-make_durationFreqCont(rangesCont)
likFun<-constrainParPaleo(likFun,q.1~0.1)
optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
    method="L-BFGS-B",control=list(maxit=1000000))

#actually decreases our sampling rate estimate
   # gets further away from true generating value, r = 0.5 (geesh!)
   # but this *is* a small dataset...
```

equation2function — *Turn a Character String of the Right-Hand Side of an Equation into an R Function*

Description

equation2function converts the right-hand side of an equation that can be written as a single line (like the right-hand side of an object of class formula) and creates an R function which calls the variables within as arguments and returns values consistent with the parameters of the input equation as written.

Usage

```
equation2function(equation, envir = parent.frame(), notName = "XXXXXXXXXXX")
```

Arguments

equation	The right-hand-side (RHS) of an equation, given as a character string. If not of type character, equation2function attempts to coerce equation to type character and fails if it cannot.
envir	The environment the resulting function will be evaluated in. See as.function.
notName	A useless string used simply as a placeholder in turning equation into a function, which should not match any actual variable in equation. Only supplied as an argument in case any

Details

This simple little function is rather hacky but seems to get the job done, for a functionality that doesn't seem to be present elsewhere in R.

Value

A function, with named blank (i.e. no default value) arguments.

Author(s)

David W. Bapst

Examples

```
# some simple examples
foo<-equation2function("x+y")
foo
foo(x=4,y=0.1)

foo<-equation2function("x+2*sqrt(2*y+3)^2")
foo
foo(x=4,y=0.1)

# what about weird long argument names and spaces
foo<-equation2function("stegosaur + 0.4 * P")
foo
foo(stegosaur=5,P=0.3)
```

`expandTaxonTree` *Extrapolating Lower-Level Taxon Phylogenies from Higher-Level Taxon Trees*

Description

This function takes a tree composed of higher-level taxa and a vector of lower-level taxa belonging to the set of higher-level taxa included in the input tree and produces a tree composed of the lower-level taxa, by treating the higher-level taxa as unresolved monophyletic polytomies. A user can also mark higher taxa as paraphyletic such that these are secondarily collapsed and do not form monophyletic clades in the output tree.

Usage

```
expandTaxonTree(taxonTree, taxaData, collapse = NULL, keepBrLen = FALSE,
  plot = FALSE)
```

Arguments

`taxonTree`	A phylo object where tips represent higher taxa
`taxaData`	Character vector of higher taxa, with elements names equal to the lower taxa. See below.
`collapse`	Character vector of non-monophyletic higher taxa to be collapsed
`keepBrLen`	Logical, decides if branch lengths should be kept or discarded. FALSE by default. See details below.
`plot`	If true, plots a comparison between input and output trees

Details

The output tree will probably be a rough unresolved view of the relationships among the taxa, due to the treatment of higher-level taxa as polytomies. This is similar to the methods used in Webb and Donoghue (2005) and Friedman (2009). Any analyses should be done by resolving this tree with `multi2di` in the ape package or via the various time-scaling functions found in this package (paleotree).

The taxaData vector should have one element per lower-level taxon that is to be added to the tree. The name of each element in the vector should be the names of the lower-level taxa, which will be used as new tip labels of the output lower-taxon tree. There should be no empty elements! expandTaxonTree won't know what to do with taxa that don't go anywhere.

By default, all higher-level taxa are treated as monophyletic clades if not otherwise specified. The collapse vector can (and probably should) be used if there is doubt about the monophyly of any higher-level taxa included in the input taxon-tree, so that such a group would be treated as a paraphyletic group in the output tree.

Also by default, the output tree will lack branch lengths and thus will not be time-scaled. If keepBrLen is true, then the tree's edge lengths are kept and new taxa are added as zero length branches attaching to a node that represents the previous higher-taxon. This tree is probably not useful for most applications, and may even strongly bias some analyses. USE WITH CAUTION! The 'collapse' vector will cause such edges to be replaced by zero-length branches rather than fully collapsing them, which could have odd effects. If 'collapse' is not null and keepBrLen is true, a warning is issued that the output probably won't make much sense at all.

Value

Outputs the modified tree as an object of class phylo, with the higher-level taxa expanded into polytomies and the lower-level taxa as the tip labels.

Author(s)

David W. Bapst

References

Friedman, M. 2009 Ecomorphological selectivity among marine teleost fishes during the end-Cretaceous extinction. *Proceedings of the National Academy of Sciences* **106**(13):5218–5223.

Webb, C. O., and M. J. Donoghue. 2005 Phylomatic: tree assembly for applied phylogenetics. *Molecular Ecology Notes* **5**(1):181–183.

See Also

`multi2di`, `bind.tree`

Examples

```
set.seed(444)
#lets make our hypothetical simulated tree of higher taxa
taxtr <- rtree(10)
```

```
taxd <- sample(taxtr$tip.label,30,replace=TRUE) #taxa to place within higher taxa
names(taxd) <- paste(taxd,"_x",1:30,sep="")
coll <- sample(taxtr$tip.label,3) #what to collapse?
expandTaxonTree(taxonTree=taxtr,taxaData=taxd,collapse=coll,plot=TRUE)
```

`footeValues` *Calculates Values for Foote's Inverse Survivorship Analyses*

Description

This function calculates the intermediary values needed for fitting Foote's inverse survivorship analyses, as listed in the table of equations in Foote (2003), with the analyses themselves described further in Foote (2001) and Foote (2005).

Usage

```
footeValues(p, q, r, PA_n = 0, PB_1 = 0, p_cont = TRUE, q_cont = TRUE,
  Nb = 1)
```

Arguments

p	Instantaneous origination/branching rate of taxa. Under a continuous model, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Under a pulsed mode (p_cont=FALSE), p is a per-interval 'rate' which can exceed 1 (because diversity can more than double; Foote, 2003a). Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as q and r.
q	Instantaneous extinction rate of taxa. Under a continuous model, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Under a pulsed mode (q_cont=FALSE), q is a per-interval 'rate' but which cannot be observed to exceed 1 (because you can't have more taxa go extinct than exist). Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as p and r.
r	Instantaneous sampling rate of taxa, assumed to be *per interval*, or equal to the product of interval lengths and the rates per lineage time units for each interval. Given as a vector with length equal to the number of intervals, so a different value may be given for each separate interval. Must be the same length as p and q.
PA_n	The probability of sampling a taxon after the last interval included in a survivorship study. Usually zero for extinct groups, although more logically has the value of 1 when there are still extant taxa (i.e., if the last interval is the Holocene and the group is still alive, the probability of sampling them later is probably 1...). Should be a value of 0 to 1.

`PB_1`	The probability of sampling a taxon before the first interval included in a survivorship study. Should be a value of 0 to 1.
`p_cont`	If TRUE (the default), then origination is assumed to be a continuous time process with an instantaneous rate. If FALSE, the origination is treated as a pulsed discrete-time process with a probability.
`q_cont`	If TRUE (the default), then extinction is assumed to be a continuous time process with an instantaneous rate. If FALSE, the extinction is treated as a pulsed discrete-time process with a probability.
`Nb`	The number of taxa that enter an interval (b is for 'bottom'). This is an arbitrary constant used to scale other values in these calculations and can be safely set to 1.

Details

Although most calculations in this function agree with the errata for Foote's 2003 table (see references), there were some additional corrections for Prob(D|FL) made as part of a personal communication in 2013 between the package author and Michael Foote.

Value

Returns a matrix with number of rows equal to the number of intervals (i.e. the length of p, q and r) and named columns representing the different values calculated by the function: "Nb", "Nbt", "NbL", "NFt", "NFL", "PD_bt", "PD_bL", "PD_Ft", "PD_FL", "PA", "PB", "Xbt", "XbL", "XFt" and "XFL".

Author(s)

David W. Bapst, with advice from Michael Foote.

References

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Foote, M. 2003a. Origination and Extinction through the Phanerozoic: A New Approach. *The Journal of Geology* 111(2):125-148.

Foote, M. 2003b. Erratum: Origination and Extinction through the Phanerozoic: a New Approach. *The Journal of Geology* 111(6):752-753.

Foote, M. 2005. Pulsed origination and extinction in the marine realm. *Paleobiology* 31(1):6-20.

Examples

```
#very simple example with three intervals, same value for all parameters

#example rates (for the most part)
rate<-rep(0.1,3)
#all continuous
footeValues(rate,rate,rate)
#origination pulsed
```

```
footeValues(rate,rate,rate,p_cont=FALSE)
#extinction pulsed
footeValues(rate,rate,rate,q_cont=FALSE)
#all pulsed
footeValues(rate,rate,rate,p_cont=FALSE,q_cont=FALSE)
```

---

freqRat *Frequency Ratio Method for Estimating Sampling Probability*

Description

Estimate per-interval sampling probability in the fossil record from a set of discrete-interval taxon ranges using the frequency-ratio method described by Foote and Raup (1996). Can also calculate extinction rate per interval from the same data distribution.

Usage

```
freqRat(timeData, calcExtinction = FALSE, plot = FALSE)
```

Arguments

`timeData`	A 2 column matrix with the first and last occurrences of taxa given in relative time intervals. If a list of length two is given for timeData, such as would be expected if the output of binTimeData was directly input, the second element is used.
`calcExtinction`	If TRUE, the per-interval, per-lineage extinction rate is estimated as the negative slope of the log frequencies, ignoring single hits (as described in Foote and Raup, 1996.)
`plot`	If true, the histogram of observed taxon ranges is plotted, with frequencies on a linear scale

Details

This function uses the frequency ratio ("freqRat") method of Foote and Raup (1996) to estimate the per-interval sampling rate for a set of taxa. This method assumes that intervals are of fairly similar length and that taxonomic extinction and sampling works similar to homogenous Poisson processes. These analyses are ideally applied to data from single stratigraphic section but potentially are applicable to regional or global datasets, although the behavior of those datasets is less well understood.

The frequency ratio is a simple relationship between the number of taxa observed only in a single time interval (also known as singletons), the number of taxa observed only in two time intervals and the number of taxa observed in three time intervals. These respective frequencies, respectively f1, f2 and f3 can then be related to the per-interval sampling probability with the following expression.

$Sampling.Probability = (f2^2)/(f1 * f3)$

This frequency ratio is generally referred to as the 'freqRat' in paleobiological literature.

It is relatively easy to visually test if range data fits expectation that true taxon durations are exponentially distributed by plotting the frequencies of the observed ranges on a log scale: data beyond the 'singletons' category should have a linear slope, implying that durations were originally exponentially distributed. (Note, a linear scale is used for the plotting diagram made by this function when 'plot' is TRUE, so that plot cannot be used for this purpose.)

The accuracy of this method tends to be poor at small interval length and even relatively large sample sizes. A portion at the bottom of the examples in the help file examine this issue in greater detail with simulations. This package author recommends using the ML method developed in Foote (1997) instead, which is usable via the function `make_durationFreqDisc`.

As extant taxa should not be included in a freqRat calculation, any taxa listed as being in a bin with start time 0 and end time 0 (and thus being extant without question) are dropped before the model fitting it performed.

Value

This function returns the per-interval sampling probability as the "freqRat", and estimates

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

See Also

Model fitting methods in `make_durationFreqDisc` and `make_durationFreqCont`. Also see conversion methods in `sProb2sRate`, `qsProb2Comp`

Examples

```
#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.1)
#Now let's use binTimeData to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length=5)

#now, get an estimate of the sampling rate (we set it to 0.5 above)
#for discrete data we can estimate the sampling probability per interval (R)
    #i.e. this is not the same thing as the instantaneous sampling rate (r)
#can use sRate2sProb to see what we would expect
```

```
sRate2sProb(r=0.1,int.length=5)
#expect R = ~0.39

#now we can apply freqRat to get sampling probability
SampProb <- freqRat(rangesDisc,plot=TRUE)
SampProb

#est. R = ~0.25
#Not wildly accurate, is it?

#can also calculate extinction rate per interval of time
freqRat(rangesDisc,calcExtinction=TRUE)

#est. ext rate = ~0.44 per interval
#5 time-unit intervals, so ~0.44 / 5 = ~0.08 per time-unite
#That's pretty close to the generating value of 0.01, used in sampleRanges

## Not run:
#################
#The following example code (which is not run by default) examines how
#the freqRat estimates vary with sample size, interval length
#and compare it to using make_durationFreqDisc

#how good is the freqRat at 20 sampled taxa on avg?
set.seed(444)
r<-runif(100)
int.length=1
R<-sapply(r,sRate2sProb,int.length=1) #estimate R from r, assuming stuff like p=q
ntaxa<-freqRats<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(15,25), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
freqRats[i]<-freqRat(timeList)
}
plot(R,freqRats);abline(0,1)
#without the gigantic artifacts bigger than 1...
plot(R,freqRats,ylim=c(0,1));abline(0,1)
#very worrisome lookin'!

#how good is it at 100 sampled taxa on average?
set.seed(444)
r<-runif(100)
int.length=1
R<-sapply(r,sRate2sProb,int.length=1)
ntaxa<-freqRats<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(80,150), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
freqRats[i]<-freqRat(timeList)
}
plot(R,freqRats,ylim=c(0,1));abline(0,1)
#not so hot, eh?

################
#LETS CHANGE THE TIME BIN LENGTH!

#how good is it at 100 sampled taxa on average, with longer time bins?
set.seed(444)
r<-runif(100)
int.length<-10
R<-sapply(r,sRate2sProb,int.length=int.length)
ntaxa<-freqRats<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(80,150), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
freqRats[i]<-freqRat(timeList)
}
plot(R,freqRats,ylim=c(0,1));abline(0,1)
#things get more accurate as interval length increases... odd, eh?

#how good is it at 20 sampled taxa on average, with longer time bins?
set.seed(444)
r<-runif(100)
int.length<-10
R<-sapply(r,sRate2sProb,int.length=int.length)
ntaxa<-freqRats<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(15,25), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
freqRats[i]<-freqRat(timeList)
}
plot(R,freqRats,ylim=c(0,1));abline(0,1)
#still not so hot at low sample sizes, even with longer bins

########################
#ML METHOD

#how good is the ML method at 20 taxa, 1 time-unit bins?
set.seed(444)
r<-runif(100)
```

```
int.length<-1
R<-sapply(r,sRate2sProb,int.length=int.length)
ntaxa<-ML_sampProb<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(15,25), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
 likFun<-make_durationFreqDisc(timeList)
 ML_sampProb[i]<-optim(parInit(likFun),likFun,
lower=parLower(likFun),upper=parUpper(likFun),
     method="L-BFGS-B",control=list(maxit=1000000))[[1]][2]
}
plot(R,ML_sampProb);abline(0,1)
# Not so great due to likelihood surface ridges
 # but it returns values between 0-1

#how good is the ML method at 100 taxa, 1 time-unit bins?
set.seed(444)
r<-runif(100)
int.length<-1
R<-sapply(r,sRate2sProb,int.length=int.length)
ntaxa<-ML_sampProb<-numeric()
for(i in 1:length(r)){
#assuming budding model
record<-simFossilRecord(p=0.1, q=0.1, r=r[i], nruns=1,
nSamp=c(80,150), nExtant=0, plot=TRUE)
ranges<-fossilRecord2fossilRanges(record)
timeList<-binTimeData(ranges,int.length=int.length)
ntaxa[i]<-nrow(timeList[[2]])
 likFun<-make_durationFreqDisc(timeList)
 ML_sampProb[i]<-optim(parInit(likFun),likFun,
lower=parLower(likFun),upper=parUpper(likFun),
     method="L-BFGS-B",control=list(maxit=1000000))[[1]][2]
}
plot(R,ML_sampProb);abline(0,1)
#Oh, fairly nice, although still a biased uptick as R gets larger


## End(Not run)
```

graptDisparity *Morphlogical Character and Range Data for late Ordovician and Early Silurian Graptoloidea*

Description

This dataset contains a morphological character matrix (containing both a set of 45 discrete characters, and 4 continuous characters coded as minimum and maximum range values), along with biostratigraphic range data for 183 graptoloid species-level taxa from Bapst et al. (2012, PNAS). Also includes a pre-calculated distance matrix based on the character matrix, using the algorithm applied by Bapst et al (2012). Interval dates for biostratigraphic zones is taken from Sadler et al. 2011.

Format

This dataset is composed of three objects:

graptCharMatrix A matrix composed of mixed character data and a group code for 183 graptoloid taxa, with rows named with species names and columns named with character names.

graptRanges A list containing two matrices: the first matrix describes the first and last interval times for 20 graptolite biozones and the second matrix contains the first and last appearances of 183 graptolite species in those same biozones. (In other words, graptRanges has the 'timeList' format called by some paleotree functions).

graptDistMat A 183x183 matrix of pair-wise distances (dissimilarities) for the 183 graptolite species, using the algorithm for discrete characters and min-max range values described in Bapst et al.

Details

The character matrix contains characters of two differing types with a (very) small but non-neglible amount of missing character data for some taxa. This required the use of an unconventional ad hoc distance metric for the published analysis, resulting in a (very slightly) non-Euclidean distance matrix. This breaks some assumptions of some statistical analyses or requires special corrections, such as with PCO.

Note that taxonomic data were collected only for species present within an interval defined by the base of the Uncinatus biozone (~448.57 Ma) to the end of the cyphus biozone (~439.37 Ma). Many taxa have first and last appearance dates listed in `graptRanges` which are outside of this interval (see examples).

Source

Source for stratigraphic ranges and character data:

Bapst, D. W., P. C. Bullock, M. J. Melchin, H. D. Sheets, and C. E. Mitchell. 2012. Graptoloid diversity and disparity became decoupled during the Ordovician mass extinction. *Proceedings of the National Academy of Sciences* 109(9):3428-3433.

Source for interval dates for graptolite zones:

Sadler, P. M., R. A. Cooper, and M. Melchin. 2009. High-resolution, early Paleozoic (Ordovician-Silurian) time scales. *Geological Society of America Bulletin* 121(5-6):887-906.

See Also

For more example graptolite datasets, see `retiolitinae`

This data was added mainly to provide an example dataset for `nearestNeighborDist`

Examples

```
#load data
data(graptDisparity)

#separate out two components of character matrix

#45 discrete characters
discChar <- graptCharMatrix[,1:45]

#min ranges for 4 continuous characters
cMinChar <- graptCharMatrix[,c(46,48,50,52)]
#max ranges for 4 continuous characters
cMaxChar <- graptCharMatrix[,c(47,49,51,53)]

#group (clade/paraclade) coding
groupID <- graptCharMatrix[,54]

#number of species
nspec <- nrow(graptCharMatrix)

#some plotting information from Bapst et al.'s plotting scripts
grpLabel <- c("Normalo.","Monogr.","Climaco.",
"Dicrano.","Lasiogr.","Diplogr.","Retiol.")
grpColor <- c("red","purple",colors()[257],colors()[614],
colors()[124],"blue",colors()[556])

##########

#plot diversity curve of taxa
taxicDivDisc(graptRanges)

#but the actual study interval for the data is much smaller
abline(v=448.57,lwd=3) #start of study interval
abline(v=439.37,lwd=3) #end of study interval

#plot diversity curve just for study interval
taxicDivDisc(graptRanges, timelims=c(448.57,439.37))

############

#distance matrix is given as graptDistMat
   #to calculate yourself, see code below in DoNotRun section

#now, is the diagonal zero? (it should be)
all(diag(graptDistMat)==0)

#now, is the matrix symmetric? (it should be)
isSymmetric(graptDistMat)

#can apply cluster analysis
clustRes <- hclust(as.dist(graptDistMat))
```

```
plot(clustRes,labels=FALSE)

#use ape to plot with colors at the tips
dev.new(width=15)  # for a prettier plot
plot.phylo(as.phylo(clustRes),show.tip.label=FALSE,
no.margin=TRUE,direction="upwards")
tiplabels(pch=16,col=grpColor[groupID+1])
legend("bottomright",legend=grpLabel,col=grpColor,pch=16)
dev.set(2)

#can apply PCO (use lingoes correction to account for negative values
   #resulting from non-euclidean matrix
pco_res <- pcoa(graptDistMat,correction="lingoes")

#relative corrected eigenvalues
rel_corr_eig <- pco_res$values$Rel_corr_eig
layout(1:2)
plot(rel_corr_eig)
#cumulative
plot(cumsum(rel_corr_eig))

#first few axes account for very little variance!!



#well let's look at those PCO axes anyway
layout(1)
pco_axes <- pco_res$vectors
plot(pco_axes[,1],pco_axes[,2],pch=16,col=grpColor[groupID+1],
   xlab=paste("PCO Axis 1, Rel. Corr. Eigenvalue =",round(rel_corr_eig[1],3)),
   ylab=paste("PCO Axis 2, Rel. Corr. Eigenvalue =",round(rel_corr_eig[2],3)))
legend("bottomright",legend=grpLabel,col=grpColor,pch=16,ncol=2,cex=0.8)


##########m##############

## Not run:

#calculate a distance matrix (very slow!)
#Bapst et al. calculated as # char diffs / total # of chars
   #but both calculated for only non-missing characters for both taxa
#non-identical discrete states = difference for discrete traits
#non-overlapping ranges for continuous characters = difference for cont traits

distMat<-matrix(,nspec,nspec)
rownames(distMat)<-colnames(distMat)<-rownames(graptCharMatrix)
for(i in 1:nspec){ for(j in 1:nspec){    #calculate for each pair of species
   #discrete characters
   di <- discChar[i,]     #discrete character vector for species i
   dj <- discChar[j,]     #discrete character vector for species j
   #now calculate pair-wise differences for non-missing characters
   discDiff <- (di!=dj)[!is.na(di)&!is.na(dj)] #logical vector
   #
```

```
    #continuous characters: need another for() loop
    contDiff <- numeric()
    for(ct in 1:4){
        #if they do not overlap, a min must be greater than a max value
        contDiff[ct] <- cMinChar[i,ct]>cMaxChar[j,ct] | cMinChar[j,ct]>cMaxChar[i,ct]
        }
    #remove NAs
    contDiff <- contDiff[!is.na(contDiff)]
    #combine
    totalDiff <- c(discDiff,contDiff)
    #divide total difference
    distMat[i,j] <- sum(totalDiff)/length(totalDiff)
    }}

#but is it identical to the distance matrix already provided?
identical(distMat,graptDistMat)
#ehh, numerical rounding issues...

#A somewhat speeder alternative to calculate a distance matrix
distMat<-matrix(,nspec,nspec)
rownames(distMat)<-colnames(distMat)<-rownames(graptCharMatrix)
for(i in 1:(nspec-1)){ for(j in (i+1):nspec){    #calculate for each pair of species
   #now calculate pair-wise differences for non-missing characters
   discDiff <- (discChar[i,]!=discChar[j,])[
      !is.na(discChar[i,])&!is.na(discChar[j,])] #logical vector
   #continuous characters: if they do not overlap, a min must be greater than a max value
      contDiff<-sapply(1:4,function(ct)
            cMinChar[i,ct]>cMaxChar[j,ct] | cMinChar[j,ct]>cMaxChar[i,ct])
   #remove NAs, combine, divide total difference
   distMat[i,j] <- distMat[j,i] <- sum(c(discDiff,contDiff[!is.na(contDiff)]))/length(
        c(discDiff,contDiff[!is.na(contDiff)]))
   }}
diag(distMat)<-0

#but is it identical to the distance matrix already provided?
identical(distMat,graptDistMat)
#ehh, MORE numerical rounding issues...


## End(Not run)
```

`graptPBDB` *Example Occurrence and Taxonomic Datasets of the Graptolithina from the Paleobiology Database*

Description

Example datasets consisting of (a) occurrence data and (b) taxonomic data downloaded from the Paleobiology Database API for the Graptolithina.

Format

The example occurrence dataset (`graptOccPBDB`) is a data.frame consisting of 5900 occurrences (rows) and 35 variables (columns). The example taxonomy dataset (`graptTaxaPBDB`) is a data.frame consisting of 364 formal taxa (rows) and 53 variables (columns). Variables are coded in the 'pbdb' vocabulary of the PBDB API v1.2.

Details

This example PBDB data is included here for testing functions involving occurrence data and taxonomy in `paleotree`.

Source

See examples for the full R code used to obtain the data from the API. You can find the Paleobiology Database at http://paleobiodb.org

The occurrence data was entered by (in order of relative portion) P. Novack-Gottshall, M. Krause, M. Foote, A. Hendy, T. Hanson, M. Sommers and others. This same data was authorized mainly by A. Miller, W. Kiessling, M. Foote, A. Hendy, S. Holland, J. Sepkoski and others.

See Also

`taxonSortPBDBocc`, `occData2timeList`, `makePBDBtaxonTree`, `plotOccData`

Examples

```
## Not run:

#original code used to obtain this dataset on March 21st, 2015
# using version 1.2 of the Paleobiology Database API

# (sorry, URLs removed as they lead to the PBDB test server...)

save(graptOccPBDB,graptTaxaPBDB,file="graptPBDB.rdata")


## End(Not run)

# load archived example data
data(graptPBDB)

# let's visualize who entered the majority of the occurrence data
pie(sort(table(graptOccPBDB$enterer)))
# and now who authorized it
pie(sort(table(graptOccPBDB$authorizer)))
# I apologize for using pie charts.

# Let's look at age resolution of these occurrences
hist(graptOccPBDB$early_age-graptOccPBDB$late_age,
main="Age Resolution of Occurrences", xlab="Ma")
```

```
#distribution of taxa among taxonomic ranks
table(graptTaxaPBDB$taxon_rank)
barplot(table(graptTaxaPBDB$taxon_rank))
```

---

horizonSampRate *Estimate Sampling Rate from Sampling Horizon Data (Solow and Smith, 1997)*

Description

This function implements the exact maximum likelihood estimator for the instantaneous sampling rate from Solow and Smith (1997, Paleobiology), which is based on the relationship between the number of collections for a set of taxa and their durations (known precisely in continuous time).

Usage

```
horizonSampRate(sampOcc = NULL, durations = NULL, nCollections = NULL)
```

Arguments

`sampOcc`	A list with the number of elements equal to the number of taxa, and each element of the list being a numerical vector with the length equal to the number of collections for each taxon, and each value equal to the precise date of that fossil's time of collection. These dates do not need to be ordered. If not supplied, the elements `durations` and `nCollections` must be supplied.
`durations`	A vector of precise durations in continuous time, with the length equal to the number of taxa. If not supplied, this is calculated from `SampOcc`, which must be supplied.
`nCollections`	A vector of integers representing the number of collections for each taxon in the input durations. If not supplied this is calculated from `SampOcc`, which must be supplied.

Details

Given a dataset of taxa with a vector N, representing the number of collections for each taxon, and a vector D, giving the precise duration for each taxon, we can use the following maximum likelihood estimator from Solow and Smith (1997) to obtain the instantaneous sampling rate:

$samplingRate = (sum(N-1)^2)/(sum(D) * sum(N))$

This method is exclusively for datasets with very precisely dated horizons, such as microfossils from deep sea cores with very precise age models. The first and last appearance must be known very precisely to provide an equally precise estimate of the duration. Most datasets are not precise enough for this method, due to chronostratigraphic uncertainty. However, note that the age of individual collections other than the first and last appearance dates do not need to be known: its only the number of collections that matters.

Value

Returns the instantaneous sampling (in per lineage*time-units) as a single numerical value. Note that this is the instantaneous sampling rate and not the probability of sampling a taxon per interval.

References

Solow, A. R., and W. Smith. 1997. On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* 23(3):271-277.

See Also

Duration frequency methods (Foote and Raup, 1996; Foote, 1997) use ranges alone to estimate sampling parameters, implemented in `durationFreq`.

Also see the conversion functions for sampling parameters at `SamplingConv`.

Examples

```
#can simulate this type of data with sampleRanges
    # just set ranges.only = FALSE
#let's try a simulation example:
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
sampledOccurrences <- sampleRanges(taxa,r = 0.5,ranges.only = FALSE)

# now try with horizonSampRate
horizonSampRate(sampOcc = sampledOccurrences)

# but we could also try with the *other* inputs
   # useful because some datasets we may only have durations
   # and number of sampling events for
filtered <- sampledOccurrences[!is.na(sampledOccurrences)]
dur <- sapply(filtered,max) - sapply(filtered,min)
nCol <- sapply(filtered,length)
# supply as durations and nCollections
horizonSampRate(durations = dur, nCollections=nCol)
```

`inverseSurv` *Inverse Survivorship Models in the Fossil Record*

Description

This function replicates the model-fitting procedure for forward and reverse survivorship curve data, described by Michael Foote in a series of papers (2001, 2003a, 2003b, 2005). These methods are discrete interval taxon ranges, as are used in many other functions in paleotree (see function arguments). This function can implement the continuous time, pulsed interval and mixed models described in Foote (2003a and 2005).

Usage

```
make_inverseSurv(timeList, groups = NULL, p_cont = TRUE, q_cont = TRUE,
  PA_n = "fixed", PB_1 = 0, Nb = 1, drop.extant = TRUE)
```

Arguments

`timeList`	A 2 column matrix with the first and last occurrences of taxa given in relative time intervals (i.e. ordered from first to last). If a list of length two is given for timeData, such as would be expected if the output of binTimeData was directly input, the second element is used. See details. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in the second matrix) are automatically dropped from the timeList and from groups simultaneously. Living taxa observed in the modern day are expected to be listed as last observed in a special interval (0,0), i.e. begins and ends at 0 time. This interval is always automatically removed prior to the calculation intermediary data for fitting likelihood functions.
`groups`	Either NULL (the default) or matrix with the number of rows equal to the number of taxa and the number of columns equal to the number of 'systems' of categories for taxa. Taxonomic membership in different groups is indicated by numeric values. For example, a dataset could have a 'groups' matrix composed of a column representing thin and thick shelled taxa, coded 1 and 2 respectively, while the second column indicates whether taxa live in coastal, shelfal or deep marine settings, coded 1-3 respectively. Different combinations of groups will be treated as having independent sampling and extinction parameters in the default analysis, for example, thinly-shelled deep marine species will have separate parameters from thinly-shelled coastal species. Grouping systems could also represent temporal heterogeneity, for example, categorizing Paleozoic versus Mesozoic taxa. If groups are NULL (the default), all taxa are assumed to be of the same group with the same parameters. Unsampled taxa (e.g. from a simulation of sampling in the fossil record, listed as NAs in timeData or timeList) are automatically dropped from groupings and the time dataset (either timeData or timeList) and from groups simultaneously.
`p_cont`	If TRUE (the default), then origination is assumed to be a continuous time process with an instantaneous rate. If FALSE, the origination is treated as a pulsed discrete-time process with a probability.
`q_cont`	If TRUE (the default), then extinction is assumed to be a continuous time process with an instantaneous rate. If FALSE, the extinction is treated as a pulsed discrete-time process with a probability.
`PA_n`	The probability of sampling a taxon after the last interval included in a survivorship study. Usually zero for extinct groups, although more logically has the value of 1 when there are still extant taxa (i.e., if the last interval is the Holocene and the group is still alive, the probability of sampling them later is probably 1...). Should be a value of 0 to 1, NULL, or can be simply "fixed", the default option. This default "fixed" option allows make_inverseSurv to decide the value based on whether there is a modern interval (i.e. an interval that is `c(0,0)`) or not: if there is one, then `PA_n=1`, if not, then `PA_n=0`. If NULL, PA_n is treated as an additional free parameter in the output model.

`PB_1`	The probability of sampling a taxon before the first interval included in a survivorship study. Should be a value of 0 to 1, or NULL. If NULL, PB_1 is treated as an additional free parameter in the output model.
`Nb`	The number of taxa that enter an interval (b is for 'bottom'). This is an arbitrary constant used to scale other values in these calculations and can be safely set to 1.
`drop.extant`	Drops all extant taxa from a dataset, w

Details

The design of this function to handle mixed continuous and discrete time models means that parameters can mean very different things, dependent on how a model is defined. Users should carefully evaluate their function arguments and the discussion of parameter values as described in the Value section.

Value

A function of class "paleotreeFunc", which takes a vector equal to the number of parameters and returns the *negative* log likelihood (for use with optim and similar optimizing functions, which attempt to minimize support values). See the functions listed at `modelMethods` for manipulating and examining such functions and `constrainParPaleo` for constraining parameters.

The function output will take the largest number of parameters possible with respect to groupings and time-intervals, which means the number of parameters may number in the hundreds. Constraining the function for optimization is recommended except when datasets are very large.

Parameters in the output functions are named 'p', 'q' and 'r', which are respectively the origination, extinction and sampling parameters. If the respective arguments 'p_cont' and 'q_cont' are TRUE, then 'p' and 'q' will represent the instantaneous per-capita origination and extinction rates (in units of per lineage time-units). When one of these arguments is given as FALSE, the respective parameter (p or q) will represent per-lineage-interval rates. For p, this will be the per lineage-interval rate of a lineage producing another lineage (which can exceed 1 because diversity can more than double) and for q, this will be the per lineage-interval 'rate' of a lineage going extinct, which cannot be observed to exceed 1 (because the proportion of diversity that goes extinct cannot exceed 1). To obtain the per lineage-interval rates from a set of per lineage-time unit rates, simply multiply the per lineage-time-unit rate by the duration of an interval (or divide, to do the reverse; see Foote, 2003 and 2005). 'r' is always the instantaneous per-capita sampling rate, in units per lineage-time units.

If PA_n or PB_1 were given as NULL in the arguments, two additional parameters will be added, named respectively 'PA_n' and 'PB_1', and listed separately for every additional grouping. These are the probability of a taxon occurring before the first interval in the dataset (PB_1) and the probability of a taxon occurring after the last interval in a dataset (PA_n). Theses will be listed as 'PA_n.0' and 'PB_1.0' to indicate that they are not related to any particular time-interval included in the analysis, unlike the p, q, and r parameters (see below).

Groupings follow the parameter names, separated by periods; by default, the parameters will be placed in groups corresponding to the discrete intervals in the input timeList, such that make_inverseSurv will create a function with parameters 'p.1', 'q.1' and 'r.1' for interval 1; 'p.2', 'q.2' and 'r.2' for interval 2 and so on. Additional groupings given by the user are listed after this first set (e.g. 'p.1.2.2').

Calculating The Results of an Inverse Survivorship Model: Because of the complicated grouping and time interval scheme, combined with the probable preference of users to use constrained models rather that the full models, it may be difficult to infer what the rates for particular intervals and groups actually are in the final model, given the parameters that were found in the final optimization.

To account for this, the function output by `inverseSurv` also contains an alternative mode which takes input rates and returns the final values along with a rudimentary plotting function. This allows users to output per-interval and per-group parameter estimates. To select these feature, the argument `altMode` must be TRUE. This function will invisibly return the rate values for each group and interval as a list of matrices, with each matrix composed of the p, q and r rates for each interval, for a specific grouping.

This plotting is extremely primitive, and most users will probably find the invisibly returned rates to be of more interest. The function `layout` is used to play plots for different groupings in sequence, and this may lead to plots which are either hard to read or even cause errors (because of too many groupings, producing impossible plots). To repress this, the argument `plotPar` can be set to FALSE.

This capability means the function has more arguments that just the usual `par` argument that accepts the vector of parameters for running an optimization. The first of these additional arguments, `altMode` enables this alternative mode, instead of trying to estimate the negative log-likelihood from the given parameters. The other arguments augment the calculation and plotting of rates.

To summarize, a function output by inverseSurv has the following arguments:

par A vector of parameters, the same length as the number of parameters needed. For plotting, can be obtained with optimization

altMode If FALSE (the default) the function will work like ordinary model-fitting functions, returning a negative log-likelihood value for the input parameter values in `par`. If TRUE, however, the input parameters will instead be translated into the by-interval, by-group rates used for calculating the log-likelihoods, plotted (if plotPar is TRUE) and these final interval-specific rates will be returned invisibly as described above.

plotPar If TRUE (the default) the calculated rates will be plotted, with each grouping given a separate plot. This can be repressed by setting plotPar to FALSE. As the only conceivable purpose for setting plotPar to FALSE is to get the calculated rates, these will not be returned invisibly if plotPar is FALSE.

ratesPerInt If FALSE, the default option, the rates plotted and returned will be in units per lineage-time units, if those rates were being treated as rates for a continuous-time process (i.e. p_cont=TRUE and q_cont=TRUE for p and q, respectively, while r is always per lineage-time units). Otherwise, the respective rate will be in units per lineage-interval. If ratesPerInt is TRUE instead, then *all* rates, even rates modelled as continuous-time process, will be returned as per lineage-interval rates, even the sampling rate r.

logRates If FALSE (the default) rates are plotted on a linear scale. If TRUE, rates are plotted on a vertical log axis.

jitter If TRUE (default) the sampling rate and extinction rate will be plotted slightly ahead of the origination rate on the time axis, so the three rates can be easily differentiated. If false, this is repressed.

legendPoisition The position of a legend indicating which line is which of the three rates on the resulting plot. This is given as the possible positions for argument 'x' of the function `legend`, and by default is "topleft", which will be generally useful if origination and extinction rates are initially low. If legendPosition is NA, then a legend will not be plotted.

Note

This function is an entirely new rewrite of the methodology derived and presented by Foote in his studies. Thus, whether it would give identical results cannot be assumed nor is it easy to test given differences in the way data is handled between our coded functions. Furthermore, there may be differences in the math due to mistakes in the derivations caught while this function was programmed. I have tested the function by applying it to the same Sepkoski genus-level dataset that Foote used in his 2003 and 2005 papers. Users can feel free to contact me for detailed figures from this analysis. Overall, it seems my function captured the overall pattern of origination and sampling rates, at least under a model where both origination and extinction are modeled as continuous-time processes. Extinction showed considerably more variability relative to the published figures in Foote (2005). Additional analyses are being run to identify the sources of this discrepancy, and the function is being released here in paleotree on a trial basis, so that it can be more easily loaded onto remote servers. Users should be thus forewarned of this essentially 'beta' status of this function.

Author(s)

David W. Bapst, with some advice from Michael Foote.

References

Foote, M. 2001. Inferring temporal patterns of preservation, origination, and extinction from taxonomic survivorship analysis. *Paleobiology* 27(4):602-630.

Foote, M. 2003a. Origination and Extinction through the Phanerozoic: A New Approach. *The Journal of Geology* 111(2):125-148.

Foote, M. 2003b. Erratum: Origination and Extinction through the Phanerozoic: a New Approach. *The Journal of Geology* 111(6):752-753.

Foote, M. 2005. Pulsed origination and extinction in the marine realm. *Paleobiology* 31(1):6-20.

See Also

This function extensively relies on `footeValues`.

A similar format for likelihood models can be seen in `durationFreq`.

Also see `freqRat`, `sRate2sProb`, `qsRate2Comp` `sProb2sRate` and `qsProb2Comp`.

For translating between sampling probabilities and sampling rates, see `SamplingConv`.

Examples

```
# let's simulate some taxon ranges from an imperfectly sampled fossil record
set.seed(444)
record <- simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa <- fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r=0.5)
#bin the ranges into discrete time intervals
rangesDisc <- binTimeData(rangesCont,int.length=5)
```

```
#apply make_inverseSurv
likFun<-make_inverseSurv(rangesDisc)
#use constrainParPaleo to make the model time-homogenous
   #match.all~match.all will match parameters so only 2 pars: p=q and r
constrFun<-constrainParPaleo(likFun,match.all~match.all)
results <- optim(parInit(constrFun), constrFun,
      lower=parLower(constrFun), upper=parUpper(constrFun),
      method="L-BFGS-B", control=list(maxit=1000000))
results

#plot the results
constrFun(results$par, altMode=TRUE)

#unconstrained function with ALL of 225 parameters!!!
    # this will take forever to converge, so it isn't run
optim(parInit(likFun),likFun,lower=parLower(likFun),upper=parUpper(likFun),
      method="L-BFGS-B",control=list(maxit=1000000))
```

kanto *Example Species Abundances Tables*

Description

A totally fictional example of species abundance data, for testing functions that require a site-by-taxon table of community ecology data.

Format

A table of type integer, representing terrestrial fauna and flora abundance counts.

Details

A classic dataset of ecological data collected by Satoshi and Okido, consisting of individual counts for 54 terrestrial faunal and floral species, fron 23 sites across the mainland Kanto region.

Different ontogenetic stages were compounded and recorded by the common name for the first ontogenetic stage, with some inconsistency for species whose earliest stage have only been recently recognized. When separate names are commonly applied to sexual dimorphic forms, these were also combined and a single common name was used.

Note: This data is a totally made-up, satirical homage to a well-known video game series (thus constituting fair-use).

Source

Pokemon And All Respective Names are Trademark and Copyright of Nintendo 1996-2015.

Examples

```
data(kanto)

#visualize site abundances as barplots
barplotAbund<-function(x){
x<-x[,colSums(x)>0]
layout(1:(nrow(x)+1))
xpar<-par(mar=c(0,7,2,0))
for(i in 1:(nrow(x)-1)){
barplot(x[i,],ylab=rownames(x)[i],
names.arg="")
}
barplot(x[nrow(x),],
ylab=rownames(x)[nrow(x)],las=3)
par(xpar)
layout(1)
mtext("Abundances",side=2,line=3,adj=0.8)
}

#first five sites
kanto5<-kanto[1:5,]
barplotAbund(kanto5)

#get pairwise Spearman rho coefficients
rhoCoeff<-pairwiseSpearmanRho(kanto,dropAbsent="bothAbsent")

#what are the nearest-neighbor rhos (largest rho correlations)?
diag(rhoCoeff)<-NA
rhoNearest<-apply(rhoCoeff,1,max,na.rm=TRUE)
rhoNearest

# We can see the power plant sample is extremely different from the rest

# measure evenness: Hurlbert's PIE

kantoPIE<-HurlbertPIE(kanto)

# compare to dominance (relative abundance of most abundant taxon)

dominance<-apply(kanto,1,function(x) max(x)/sum(x) )

plot(kantoPIE,dominance)

# relatively strong relationship!


## Not run:

#get bray-curtis distances
library(vegan)
bcDist <- vegdist(kanto,method="bray")
```

```
#do a PCO on the bray-curtis distances
pcoRes <- pcoa(bcDist,correction="lingoes")
scores <- pcoRes$vectors
#plot the PCO
plot(scores,type="n")
text(labels=rownames(kanto),scores[,1],scores[,2],cex=0.5)

#the way the power plant the the pokemon tower converge
# is very suspicious: may be distortion due to a long gradient

#do a DCA instead with vegan's decorana
dcaRes<-decorana(kanto)
#plot using native vegan functions
#will show species scores in red
plot(dcaRes,cex=0.5)
#kind of messy

#show just the sites scores
plot(dcaRes,cex=0.5,display="sites")

#show just the species scores
plot(dcaRes,cex=0.5,display="species")

#well, that's pretty cool


#get the nearest neighbor for each site based on pair-wise rho coefficients
rhoNeighbor<-apply(rhoCoeff,1,function(x)
rownames(kanto)[tail(order(x,na.last=NA),1)])

#let's plot the nearest neighbor connections with igraph
NNtable<-cbind(rownames(kanto),rhoNeighbor)

# now plot with igraph
library(igraph)
NNlist <- graph.data.frame(NNtable)
plot(NNlist)

#arrows point at the nearest neighbor of each sample
# based on maximum Spearman rho correlation

#testing for differences between groups of sites

#is there a difference between routes and non-routes
groups<-rep(0,nrow(kanto))
groups[grep(rownames(kanto),pattern="Route")]<-1

#anosim (in vegan)
#are distances within groups smaller than distances between?
#we could also use adonis from vegan instead
library(vegan)
```

```
anosim(dat=kanto,grouping=groups)
adonis(kanto~factor(groups))
#both are very significant

#alternative: using multivariate GLMs in mvabund

library(mvabund)

ft <- manyglm(formula=kanto~factor(groups))
anova(ft)
#also highly significant!



## End(Not run)
```

---

macroperforateForam *Ancestor-Descendant Relationships for Macroperforate Foraminifera, from Aze et al. (2011)*

Description

An example dataset of ancestor-descendent relationships and first and last appearance dates for a set of macroperforate Foramanifera, taken from the supplemental materials of Aze et al. (2011). This dataset is included here primarily for testing functions `parentChild2taxonTree` and `taxa2phylo`.

Format

The 'foramAM' and 'foramAL' tables include budding taxon units for morphospecies and lineages respective, with four columns: taxon name, ancestral taxon's name, first appearance date and last appearance date (note that column headings vary). The 'foramAMb' and 'foramALb' tables are composed of data for the same taxon units as the previous branching events are split so that the relationships are fully 'bifurcating', rather than 'budding'. As this obscures taxonomic identity, taxon identification labels are included in an additional, fifth column in these tables. See the examples section for more details.

Details

This example dataset is composed of four tables, each containing information on the ancestor-descendant relationships and first and last appearances of species of macroperforate foraminifera species from the fossil record. Each of the four tables are for the same set of taxa, but divide and concatanate the included foram species in four different ways, relating to the use of morpospecies versus combined anagenetic lineages (see Ezard et al., 2012), and whether taxa are retained as units related by budding-cladogensis or the splitting of taxa at branching points to create a fully 'bifurcating' set of relationships, independent of ancestral morphotaxon persistance through branching events. See the examples section for more details.

Source

This dataset is obtained from the supplementary materials of, specifically 'Appendix S5':

Aze, T., T. H. G. Ezard, A. Purvis, H. K. Coxall, D. R. M. Stewart, B. S. Wade, and P. N. Pearson. 2011. A phylogeny of Cenozoic macroperforate planktonic foraminifera from fossil data. *Biological Reviews* 86(4):900-927.

References

This dataset has been used or referenced in a number of works, including:

Aze, T., T. H. G. Ezard, A. Purvis, H. K. Coxall, D. R. M. Stewart, B. S. Wade, and P. N. Pearson. 2013. Identifying anagenesis and cladogenesis in the fossil record. *Proceedings of the National Academy of Sciences* 110(32):E2946-E2946.

Ezard, T. H. G., T. Aze, P. N. Pearson, and A. Purvis. 2011. Interplay Between Changing Climate and Species' Ecology Drives Macroevolutionary Dynamics. *Science* 332(6027):349-351.

Ezard, T. H. G., P. N. Pearson, T. Aze, and A. Purvis. 2012. The meaning of birth and death (in macroevolutionary birth-death models). *Biology Letters* 8(1):139-142.

Ezard, T. H. G., G. H. Thomas, and A. Purvis. 2013. Inclusion of a near-complete fossil record reveals speciation-related molecular evolution. *Methods in Ecology and Evolution* 4(8):745-753.

Strotz, L. C., and A. P. Allen. 2013. Assessing the role of cladogenesis in macroevolution by integrating fossil and molecular evidence. *Proceedings of the National Academy of Sciences* 110(8):2904-2909.

Strotz, L. C., and A. P. Allen. 2013. Reply to Aze et al.: Distinguishing speciation modes based on multiple lines of evidence. *Proceedings of the National Academy of Sciences* 110(32):E2947-E2947.

Examples

```
# Following Text Reproduced from Aze et al. 2011's Supplemental Material
# Appendix S5
#
# 'Data required to produce all of the phylogenies included in the manuscript
# using paleoPhylo (Ezard & Purvis, 2009) a free software package to draw
# paleobiological phylogenies in R.'
#
# 'The four tabs hold different versions of our phylogeny:
#  aMb: fully bifurcating morphospecies phylogeny
#  aM: budding/bifurcating morphospecies phylogeny
#  aLb: fully bifurcating lineage phylogeny
#  aL: budding/bifurcating lineage phylogeny
#
# 'Start Date gives the first occurence of the species according
# to the particular phylogeny; End Date gives the last occurence
# according to the particular phylogeny.'

## Not run:

#load the data
```

```
#given in supplemental as XLS sheets
#converted to separate tab-deliminated text files

# aM: budding/bifurcating morphospecies phylogeny
foramAM<-read.table(file.choose(),stringsAsFactors=FALSE,header=TRUE)
# aL: budding/bifurcating lineage phylogeny
foramAL<-read.table(file.choose(),stringsAsFactors=FALSE,header=TRUE)
# aMb: fully bifurcating morphospecies phylogeny
foramAMb<-read.table(file.choose(),stringsAsFactors=FALSE,header=TRUE)
# aLb: fully bifurcating lineage phylogeny
foramALb<-read.table(file.choose(),stringsAsFactors=FALSE,header=TRUE)

save.image("macroperforateForam.rdata")


## End(Not run)

#instead, we'll just load the data directly
data(macroperforateForam)

#Two distinctions among the four datasets:
#(1): morphospecies vs morphospecies combined into sequences of anagenetic
# morpospecies referred to as 'lineages'. Thus far more morphospecies
# than lineages. The names of lineages are given as the sequence of
# their respective component morphospecies.
#(2): Datasets where taxon units (morphospecies or lineages) are broken up
# at 'budding' branching events (where the ancestral taxon persists)
# so that final dataset is 'fully bifurcating', presumably
# to make comparison easier to extant-taxon only datasets.
# (This isn't a limitation for paleotree, though!).
# This division of taxon units requires abstracting the taxon IDs,
# requiring another column for Species Name.

dim(foramAM)
dim(foramAL)
dim(foramAMb)
dim(foramALb)

#Need to convert these to same format as fossilRecord2fossilTaxa output.
#those 'taxa' tables has 6 columns:
#taxon.id ancestor.id orig.time ext.time still.alive looks.like

#for the purposes of this, we'll make taxon.id=looks.like
# (That's only for simulating cryptic speciation anyway)
#still.alive should be TRUE (1) if ext.time=0

#a function to convert Aze et al's suppmat to paleotree-readable format

createTaxaData<-function(table){
#reorder table by first appearance time
table<-table[order(-as.numeric(table[,3])),]
ID<-1:nrow(table)
anc<-sapply(table[,2],function(x)
if(!is.na(x)){
which(x==table[,1])
}else{ NA })
stillAlive<-as.numeric(table[,4]==0)
ages<-cbind(as.numeric(table[,3]),as.numeric(table[,4]))
res<-cbind(ID,anc,ages,stillAlive,ID)
colnames(res)<-c('taxon.id','ancestor.id','orig.time',
'ext.time','still.alive','looks.like')
rownames(res)<-table[,1]
return(res)
}

taxaAM<-createTaxaData(foramAM)
taxaAMb<-createTaxaData(foramAMb)
taxaAL<-createTaxaData(foramAL)
taxaALb<-createTaxaData(foramALb)

##################################

#Checking Ancestor-Descendant Relationships for Irregularities

#For each of these, there should only be a single taxon
# without a parent listed (essentially, the root ancestor)

countParentsWithoutMatch<-function(table){
     parentMatch<-match(unique(table[,2]),table[,1])
     sum(is.na(parentMatch))
}

#test this on the provided ancestor-descendant relationships
countParentsWithoutMatch(foramAM)
countParentsWithoutMatch(foramAL)
countParentsWithoutMatch(foramAMb)
countParentsWithoutMatch(foramALb)

#and on the converted datasets
countParentsWithoutMatch(taxaAM)
countParentsWithoutMatch(taxaAL)
countParentsWithoutMatch(taxaAMb)
countParentsWithoutMatch(taxaALb)



#can construct the parentChild2taxonTree
#using the ancestor-descendant relationships

#can be very slow...

treeAM<-parentChild2taxonTree(foramAM[,2:1])
treeAL<-parentChild2taxonTree(foramAL[,2:1])
treeAMb<-parentChild2taxonTree(foramAMb[,2:1])
treeALb<-parentChild2taxonTree(foramALb[,2:1])
```

```
layout(matrix(1:4,2,2))
plot(treeAM,main='treeAM',show.tip.label=FALSE)
plot(treeAL,main='treeAL',show.tip.label=FALSE)
plot(treeAMb,main='treeAMb',show.tip.label=FALSE)
plot(treeALb,main='treeALb',show.tip.label=FALSE)

# FYI
# in case you were wondering
# you would *not* time-scale these Frankenstein monsters



###########################################

# Checking stratigraphic ranges

# do all first occurrence dates occur before last occurrence dates?
# we'll check the original datasets here

checkFoLo<-function(data){
diffDate<-data[,3]-data[,4] #subtract LO from FO
isGood<-all(diffDate>=0) #is it good
return(isGood)
}

checkFoLo(foramAM)
checkFoLo(foramAL)
checkFoLo(foramAMb)
checkFoLo(foramALb)

#cool, but do all ancestors appear before their descendents?
# easier to check unified fossilRecord2fossilTaxa format here

checkAncOrder<-function(taxa){
#get ancestor's first occurrence
ancFO<-taxa[taxa[,2],3]
#get descendant's first occurrence
descFO<-taxa[,3]
diffDate<-ancFO-descFO #subtract descFO from ancFO
#remove NAs due to root taxon
diffDate<-diffDate[!is.na(diffDate)]
isGood<-all(diffDate>=0) #is it all good
return(isGood)
}

checkAncOrder(taxaAM)
checkAncOrder(taxaAL)
checkAncOrder(taxaAMb)
checkAncOrder(taxaALb)

#now, are there gaps between the last occurrence of ancestors
# and the first occurrence of descendents?
# (shall we call these 'stratophenetic ghost branches'?!)
# These shouldn't be problematic, but do they occur in this data?
# After all, fossilRecord2fossilTaxa output tables are designed for
   # fully observed simulated fossil records with no gaps.

sumAncDescGap<-function(taxa){
#get ancestor's last occurrence
ancLO<-taxa[taxa[,2],4]
#get descendant's first occurrence
descFO<-taxa[,3]
diffDate<-ancLO-descFO #subtract descFO from ancFO
#remove NAs due to root taxon
diffDate<-diffDate[!is.na(diffDate)]
#should be negative or zero, positive values are gaps
gaps<-c(0,diffDate[diffDate>0])
sumGap<-sum(gaps)
return(sumGap)
}

#get the total gap between ancestor LO and child FO
sumAncDescGap(taxaAM)
sumAncDescGap(taxaAL)
sumAncDescGap(taxaAMb)
sumAncDescGap(taxaALb)

#It appears there is *no* gaps between ancestors and their descendants
#in the Aze et al. foram dataset... wow!

###############



# Creating time-scaled phylogenies from the Aze et al. data

# Aze et al. (2011) defines anagenesis such that taxa may overlap
# in time during a transitional period (see Ezard et al. 2012
# for discussion of this definition). Thus, we would expect that
# paleotree obtains very different trees for morphospecies versus
# lineages, but very similar phylogenies for datasets where budding
# taxa are retained or arbitrarily broken into bifurcating units.

# We can use the function taxa2phylo to directly create
# time-scaled phylogenies from the Aze et al. stratophenetic data

timetreeAM<-taxa2phylo(taxaAM)
timetreeAL<-taxa2phylo(taxaAL)
timetreeAMb<-taxa2phylo(taxaAMb)
timetreeALb<-taxa2phylo(taxaALb)

layout(matrix(1:4,2,2))
plot(timetreeAM,main='timetreeAM',show.tip.label=FALSE)
axisPhylo()
plot(timetreeAL,main='timetreeAL',show.tip.label=FALSE)
axisPhylo()
```

```
plot(timetreeAMb,main='timetreeAMb',show.tip.label=FALSE)
axisPhylo()
plot(timetreeALb,main='timetreeALb',show.tip.label=FALSE)
axisPhylo()

#visually compare the two pairs we expect to be close to identical

#morpospecies
layout(1:2)
plot(timetreeAM,main='timetreeAM',show.tip.label=FALSE)
axisPhylo()
plot(timetreeAMb,main='timetreeAMb',show.tip.label=FALSE)
axisPhylo()

#lineages
layout(1:2)
plot(timetreeAL,main='timetreeAL',show.tip.label=FALSE)
axisPhylo()
plot(timetreeALb,main='timetreeALb',show.tip.label=FALSE)
axisPhylo()

layout(1)

#compare the summary statistics of the trees
Ntip(timetreeAM)
Ntip(timetreeAMb)
Ntip(timetreeAL)
Ntip(timetreeALb)
# very different!

# after dropping anagenetic zero-length-terminal-edge ancestors
# we would expect morphospecies and lineage phylogenies to be very similar

#morphospecies
Ntip(dropZLB(timetreeAM))
Ntip(dropZLB(timetreeAMb))
#identical!

#lineages
Ntip(dropZLB(timetreeAL))
Ntip(dropZLB(timetreeALb))
# ah, very close, off by a single tip
# ...probably a very short ZLB outside tolerance

#we can create some diversity plots to compare

multiDiv(data=list(timetreeAM,timetreeAMb),
plotMultCurves=TRUE)

multiDiv(data=list(timetreeAL,timetreeALb),
plotMultCurves=TRUE)

# we can see that the morphospecies datasets are identical
# that's why we can only see one line
# some very slight disagreement between the lineage datasets
# around ~30-20 Ma

#can also compare morphospecies and lineages diversity curves

multiDiv(data=list(timetreeAM,timetreeAL),
plotMultCurves=TRUE)

#they are similar, but some peaks are missing from lineages
# particularly around ~20-10 Ma
```

---

makePBDBtaxonTree — *Creating a Taxon-Tree from Taxonomic Data Downloaded from the Paleobiology Database*

Description

This function creates phylogeny-like object of type `phylo` from the taxonomic information recorded in a taxonomy download from the PBDB for a given group. Two different algorithms are provided, the default being based on parent-child taxon relationships, the other based on the nested Linnean hierarchy.

Usage

```
makePBDBtaxonTree(data, rank, method = "parentChild", solveMissing = NULL,
  tipSet = "nonParents", cleanTree = TRUE, APIversion = "1.1")
```

Arguments

`data`	A table of taxonomic data collected from the Paleobiology Database, using the taxa list option with show=phylo. Should work with versions 1.1-1.2 of the API, with either the 'pbdb' or 'com' vocab. However, as 'accepted_name' is not available in API v1.1, the resulting tree will have a taxon's *original* name and not any formally updated name.
`rank`	The selected taxon rank; must be one of 'species', 'genus', 'family', 'order', 'class' or 'phylum'.
`method`	Controls which algorithm is used for calculating the taxon-tree: either `method = "parentChild"` (the default option) which converts the listed binary parent-child taxon relationships from the input PBDB data, or `method = "Linnean"`, which converts a taxon-tree by creating a table of the Linnean taxonomic assignments (family, order, etc), which are provided when option 'show=phylo' is used in PBDB API calls.

`solveMissing`	Under `method = "parentChild"`, what should `makePBDBtaxonTree` do about multiple 'floating' parent taxa, listed without their own parent taxon information in the input dataset under `data`? Each of these is essentially a separate root taxon, for a different set of parent-child relationships, and thus poses a problem as far as returning a single phylogeny is concerned. If `solveMissing = NULL` (the default), nothing is done and the operation halts with an error, reporting the identity of these taxa. Two alternative solutions are offered: first, `solveMissing = "mergeRoots"` will combine these disparate potential roots and link them to an artificially-constructed pseudo-root, which at least allows for visualization of the taxonomic structure in a limited dataset. Secondly, `solveMissing = "queryPBDB"` queries the Paleobiology Database repeatedly via the API for information on parent taxa of the 'floating' parents, and continues within a `while()` loop until only one such unassigned parent taxon remains. This latter option may talk a long time or never finish, depending on the linearity and taxonomic structures encountered in the PBDB taxonomic data; i.e. if someone a taxon was ultimately its own indirect child in some grand loop by mistake, then under this option `makePBDBtaxonTree` might never finish. In cases where taxonomy is bad due to weird and erroneous taxonomic assignments reported by the PBDB, this routine may search all the way back to a very ancient and deep taxon, such as the Eukaryota taxon. Users should thus use `solveMissing = "queryPBDB"` only with caution.
`tipSet`	This argument only impacts analyses where the argument `method = "parentChild"` is also used. This `tipSet` controls which taxa are selected as tip taxa for the output tree. The default `tipSet = "nonParents"` selects all child taxa which are not listed as parents in `parentChild`. Alternatively, `tipSet = "all"` will add a tip to every internal node with the parent-taxon name encapsulated in parentheses.
`cleanTree`	By default, the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If FALSE, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.
`APIversion`	Version of the Paleobiology Database API used by `makePBDBtaxonTree` when `solveMissing = "queryPBDB"`. The current default is "1.1", which is the only option available as of 05/05/2015. In the future, the improved API version "1.2" will be released on the public PBDB server, which will become the new default for this function, but the option to return to "1.1" behavior will be retained for .

Details

This function should not be taken too seriously. Many groups in the Paleobiology Database have out-of-date or very incomplete taxonomic information. This function is meant to help visualize what information is present, and by use of time-scaling functions, allow us to visualize the intersection of temporal and phylogenetic, mainly to look for incongruence due to either incorrect taxonomic placements, erroneous occurrence data or both.

Note however that, contrary to common opinion among some paleontologists, taxon-trees may be just as useful for macroevolutionary studies as reconstructed phylogenies (Soul and Friedman, in press.).

Value

A phylogeny of class 'phylo', where each tip is a taxon of the given 'rank'. See additional details regarding branch lengths can be found in the sub-algorithms used to create the taxon-tree by this function: `parentChild2taxonTree` and `taxonTable2taxonTree`.

Depending on the `method` used, either the element `$parentChild` or `$taxonTable` is added to the list structure of the output phylogeny object, which was used as input for one of the two algorithms mentioned above.

Please note that when applied to output from the taxa option of the API version 1.1, the taxon names returned are the *original* taxon names as 'accepted_name' is not available in API v1.1, while under API v1.2, the returned taxon names should be the most up-to-date formal names for those taxa. Similar issues also effect the identification of parent taxa, as the accepted name of the parent ID number is only provided in version 1.2 of the API.

Author(s)

David W. Bapst

References

Soul, L. C., and M. Friedman. In Press. Taxonomy and Phylogeny Can Yield Comparable Results in Comparative Palaeontological Analyses. *Systematic Biology* (Link)

See Also

Two other functions in paleotree are used as sub-algorithms by `makePBDBtaxonTree` to create the taxon-tree within this function, and users should consult their manual pages for additional details:

`parentChild2taxonTree` and `taxonTable2taxonTree`

Other functions for manipulating PBDB data can be found at `taxonSortPBDBocc`, `occData2timeList`, and the example data at `graptPBDB`.

Examples

```
## Not run:

easyGetPBDBtaxa<-function(taxon,show=c("phylo","img","app")){
#let's get some taxonomic data
taxaData<-read.csv(paste0("http://paleobiodb.org/",
"data1.1/taxa/list.txt?base_name=",taxon,
"&rel=all_children&show=",
paste0(show,collapse=","),"&status=senior"),
stringsAsFactors=FALSE)
return(taxaData)
}

#graptolites
graptData<-easyGetPBDBtaxa("Graptolithina")
graptTree<-makePBDBtaxonTree(graptData,"genus",
method="parentChild", solveMissing="queryPBDB")
#try Linnean
```

```
graptTree<-makePBDBtaxonTree(graptData,"genus",
method="Linnean")
plot(graptTree,show.tip.label=FALSE,no.margin=TRUE,edge.width=0.35)
nodelabels(graptTree$node.label,adj=c(0,1/2))

#conodonts
conoData<-easyGetPBDBtaxa("Conodonta")
conoTree<-makePBDBtaxonTree(conoData,"genus",
method="parentChild", solveMissing="queryPBDB")
plot(conoTree,show.tip.label=FALSE,no.margin=TRUE,edge.width=0.35)
nodelabels(conoTree$node.label,adj=c(0,1/2))

#asaphid trilobites
asaData<-easyGetPBDBtaxa("Asaphida")
asaTree<-makePBDBtaxonTree(asaData,"genus",
method="parentChild", solveMissing="queryPBDB")
plot(asaTree,show.tip.label=FALSE,no.margin=TRUE,edge.width=0.35)
nodelabels(asaTree$node.label,adj=c(0,1/2))

#Ornithischia
ornithData<-easyGetPBDBtaxa("Ornithischia")
ornithTree<-makePBDBtaxonTree(ornithData,"genus",
method="parentChild", solveMissing="queryPBDB")
#try Linnean
#need to drop repeated taxon first: Hylaeosaurus
ornithData<-ornithData[-(which(ornithData[,"taxon_name"]=="Hylaeosaurus")[1]),]
ornithTree<-makePBDBtaxonTree(ornithData,"genus",
method="Linnean")
plot(ornithTree,show.tip.label=FALSE,no.margin=TRUE,edge.width=0.35)
nodelabels(ornithTree$node.label,adj=c(0,1/2))

#Rhynchonellida
rynchData<-easyGetPBDBtaxa("Rhynchonellida")
rynchTree<-makePBDBtaxonTree(rynchData,"genus",
method="parentChild", solveMissing="queryPBDB")
plot(rynchTree,show.tip.label=FALSE,no.margin=TRUE,edge.width=0.35)
nodelabels(rynchTree$node.label,adj=c(0,1/2))

#some of these look pretty messy!


## End(Not run)

###################################


#let's try time-scaling the graptolite tree

#get some example occurrence and taxonomic data
data(graptPBDB)

#get the taxon tree: Linnean method
graptTree<-makePBDBtaxonTree(graptTaxaPBDB, "genus", method="Linnean")
```

```
plot(graptTree,cex=0.4)
nodelabels(graptTree$node.label,cex=0.5)

#get the taxon tree: parentChild method
graptTree<-makePBDBtaxonTree(graptTaxaPBDB, "genus", method="parentChild")
plot(graptTree,cex=0.4)
nodelabels(graptTree$node.label,cex=0.5)

#get time data from occurrences
graptOccGenus<-taxonSortPBDBocc(graptOccPBDB,rank="genus",onlyFormal=FALSE)
graptTimeGenus<-occData2timeList(occList=graptOccGenus)

#let's time-scale the parentChild tree with paleotree
# use minimum branch length for visualization
# and nonstoch.bin so we plot maximal ranges
timeTree<-bin_timePaleoPhy(graptTree,timeList=graptTimeGenus,
nonstoch.bin=TRUE,type="mbl",vartime=3)

#drops a lot of taxa; some of this is due to mispellings, etc


## Not run:

#make pretty plot with library strap
library(strap)
geoscalePhylo(timeTree, ages=timeTree$ranges.used)
nodelabels(timeTree$node.label,cex=0.5)


## End(Not run)
```

`minBranchLength` *Scales Edge Lengths of a Phylogeny to a Minimum Branch Length*

Description

Rescales a tree with edge lengths so that all edge lengths are at least some minimum branch length (mbl), without changing the relative distance of the tips from the root node. Edge lengths are transformed so they are greater than or equal to the input minimum branch length, by subtracting edge length from more rootward edges and added to later branches.

Usage

```
minBranchLength(tree, mbl)
```

Arguments

`tree`	A phylogeny with edge lengths of class 'phylo'.
`mbl`	The minimum branch length

Details

This function was formally an internal segment in `timePaleoPhy`, and now is called by `timePaleoPhy` instead, allowing users to apply `minBranchLength` to trees that already have edge lengths.

Value

A phylogeny with edge lengths of class 'phylo'.

Author(s)

David W. Bapst

See Also

This function was originally an internal piece of `timePaleoPhy`, which implements the minimum branch length time-scaling method along with others, which may be what you're looking for (instead of this miscellaneous function).

Examples

```
#simulation with an example non-ultrametric tree

tree<-rtree(20)
# randomly replace edges with ZLBs, similar to multi2di output
tree<-degradeTree(tree,0.3,leave.zlb=TRUE)

tree2<-minBranchLength(tree,0.1)

layout(1:2)
plot(tree);axisPhylo()
plot(tree2);axisPhylo()

layout(1)

#now let's try it with an ultrametric case

# get a random tree
tree<-rtree(30)
# randomly replace edges with ZLBs, similar to multi2di output
tree<-degradeTree(tree,0.5,leave.zlb=TRUE)
# now randomly resolve
tree<-di2multi(tree)
# give branch lengths so its ultrametric
tree<-compute.brlen(tree)

plot(tree) #and we have an ultrametric tree with polytomies, yay!

#now randomly resolve, get new branch lengths as would with real data
tree2<-multi2di(tree)
tree2<-minBranchLength(tree2,0.1)
```

```
layout(1:2)
plot(tree,show.tip.label=FALSE);axisPhylo()
plot(tree2,show.tip.label=FALSE);axisPhylo()

layout(1)
```

minCharChange *Estimating the Minimum Number of Character Transitions Using Maximum Parsimony*

Description

minCharChange is a function which takes a cladogram and a discrete trait and finds the solutions of inferred character states for ancestral nodes that minimizes the number of character state transitions (either gains or losses/reversals) for a given topology and a set of discrete character data. minCharChange relies on ancPropStateMat, which is a wrapper for phangorn's function ancestral.pars.

Usage

```
minCharChange(trait, tree, randomMax = 10000, maxParsimony = TRUE,
  orderedChar = FALSE, type = "MPR", cost = NULL, printMinResult = TRUE,
  ambiguity = c(NA, "?"), dropAmbiguity = FALSE, polySymbol = "&",
  contrast = NULL)

ancPropStateMat(trait, tree, orderedChar = FALSE, type = "MPR",
  cost = NULL, ambiguity = c(NA, "?"), dropAmbiguity = FALSE,
  polySymbol = "&", contrast = NULL, returnContrast = FALSE)
```

Arguments

trait	A vector of trait values for a discrete character, preferably named with taxon names identical to the tip labels on the input tree.
tree	A cladogram of type 'phylo'. Any branch lengths are ignored.
randomMax	The maximum number of cladograms examined when searching a large number of solutions consistent with the reconstructed ancestral states from ancestral.pars with the minimum number of character state transitions. If the number of potential solutions is less than randomMax, then solutions are exhaustively searched.
maxParsimony	If maxParsimony is TRUE (the default) then only solutions with the smallest number of total transitions examined will be returned. Note that since solutions are stochastically 'guessed' at, and the number of possible solutions may not be exhaustively searched, there may have been solutions not examined with a lower number of transitions even if maxParsimony = TRUE. Regardless, one may want to do maxParsimony = FALSE if one is interested in whether there are solutions with a smaller number of gains or losses and thus wants to return all solutions.

`orderedChar`	If TRUE (not the default), then the character will be reconstructed with a cost (step) matrix of a linear, ordered character. This is not applicable if `type = "ACCTRAN"`, as cost matrices cannot be used with ACCTRAN in `ancestral.pars`, and an error will be returned if `orderedChar = TRUE` but a cost matrix is given, as the only reason to use orderedChar is to produce a cost matrix automatically.
`type`	The parsimony algorithm applied by `ancestral.pars`, which can apply one of two: "MPR" (the default) is a relatively fast algorithm developed by Hamazawa et al. (1995) and Narushima and Hanazawa (1997), which relies on reconstructing the states at each internal node by re-rooting at that node. "ACCTRAN", the 'accelerated transitions' algorithm (Swofford and Maddison, 1987), favors character reversal over independent gains when there is ambiguity. The "ACCTRAN" option in ancestral.pars avoids repeated rerooting of the tree to search for a smaller set of maximum-parsimony solutions that satisfy the ACCTRAN algorithm, but does so by assigning edge weights. As of phangorn v1.99-12, both of these algorithms apply the Sankoff parsimony algorithm, which allows multifurcations (polytomies).
`cost`	A matrix of the cost (i.e. number of steps) necessary to change between states of the input character trait. If NULL (the default), the character is assumed to be unordered with equal cost to change from any state to another. Cost matrices only impact the "MPR" algorithm; if a cost matrix is given but `type = "ACCTRAN"`, an error is issued.
`printMinResult`	If TRUE (the default), a summary of the results is printed to the terminal. The information in this summary may be more detailed if the results of the analysis are simpler (i.e. fewer unique solutions).
`ambiguity`	A vector of values which indicate ambiguous (i.e. missing or unknown) character state codings in supplied `trait` data. Taxa coded ambiguously as treated as being equally likely to be any state coding. By default, `NA` values and "?" symbols are treated as ambiguous character codings, in agreement with behavior of functions in packages `phangorn` and `Claddis`. This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package phangorn.
`dropAmbiguity`	A logical. If `TRUE` (which is not the default), all taxa with ambiguous codings as defined by argument `ambiguity` will be dropped prior to ancestral nodes being inferred. This may result in too few taxa.
`polySymbol`	A single symbol which separates alternative states for polymorphic codings; the default symbol is `"&"`, following the output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated by default with a string with state labels separated by an `"&"` symbol. For example, a taxon coded as polymorphic for states 1 or 2, would be indicated by the string "1&2". `polySymbol` is used to break up these strings and automatically construct a fitting `contrast` table for use with this data, including for ambiguous character state codings.
`contrast`	A matrix of type integer with cells of 0 and 1, where each row is labelled with a string value used for indicating character states in `trait`, and each column is labelled with the formal state label to be used for assign taxa to particular character states. A value of 1 indicates that the respective coding string for that row should be interpreted as reflecting the character state listed for that column. A

coding could reflect multiple states (such as might occur when taxa are polymorphic for some morphological character), so the sums of rows and columns can sum to more than 1. If `contrast` is not `NULL` (the default), the arguments will nullify This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package phangorn. This structure is based on the `contrasts` tables used for statistical evaluation of factors. See the `phangorn` vignette "Special features of phangorn" for more details on its implementation within `phangorn` including an example. See examples below for the construction of an example contrast matrix for character data with polymorphisms, coded as character data output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated with a string with state labels separated by an `"&"` symbol.

`returnContrast` If TRUE, the contrast table used by `ancestral.pars` will be output instead for user evaluation that polymorphic symbols and ambiguous states are being parsed correctly.

Details

The wrapper function `ancPropStateMat` simply automates the application of functions `ancestral.pars` and `phyDat` from `phangorn`, along with several additional checks and code to present the result as a matrix, rather than a specialized list.

Note that although the default `orderedChar` argument assumes that multistate characters are unordered, the results of character change will always be reported as gains and losses relative to the numbering of the states in the output `transitionSumChanges`, exactly as if they had been ordered. In the case where the character is actually ordered, this may be considered a conservative approach, as using a parsimony algorithm for unordered character states allows fewer gains or losses to be counted on branches where multiple gains and losses are reported. If the character is presumably unordered *and multistate*, however, then the gains and losses division is *arbitrary nonsense* and should be combined to to obtain the total number of character changes.

Value

By default, `ancPropStateMat` returns a matrix, with rows corresponding to the ID numbers of tips and nodes in `$edge`, and columns corresponding to character states, with the value representing the proportional weight of that node being that state under the algorithm used (known tip values are always 1). If argument `returnContrast` is `TRUE` then `ancPropStateMat` will instead return the final contrast table used by `phyDat` for interpreting character state strings.

`minCharChange` invisibly returns a list containing the following elements, several of which are printed by default to the console, as controlled by argument `printMinResult`:

`message` Describes the performance of `minCharChange` at searching for a minimum solution.

`sumTransitions` A vector recording the total number of necessary transitions (sum total of gains and losses/reversal) for each solution; effectively the parsimony cost of each solution.

`minTransitions` A symmetrical matrix with number of rows and columns equal to the number of character states, with values in each cell indicating the minimum number of transitions from one ancestral state (i.e. the rows) to a descendant state (i.e. the columns), taken across the set of kept solutions (dependent on which are kept as decided by argument `maxParsimony`). Generally guaranteed not to add up to the number of edges contained within the input tree, and thus may not represent any realistic evolutionary scenario but does represent a conservative approach for asking 'what is the smallest possible number of transitions from 0 to 1' or 'smallest possible number of transitions from 1 to 0', independently of each other.

`solutionArray` A three-dimensional array, where for each solution, we have a matrix with edges for rows and two columns indicating the ancestral and child nodes of that edge, with values indicating the states inferred for those nodes in a particular solution.

`transitionArray` A labelled three-dimensional array where for each solution we have a symmetrical matrix with number of rows and columns equal to the number of character states, with values in each cell indicating the total number of transitions from one ancestral state (i.e. the rows) to a descendant state (i.e. the columns).

`transitionSumChanges` Which is a three column matrix with a row for every solution, with the values in the three columns measuring the number of edges (branches) inferred to respectively have gains, no change or losses (i.e. reversals), as calculated relative to the order of character states.

Author(s)

David W. Bapst

References

Hanazawa, M., H. Narushima, and N. Minaka. 1995. Generating most parsimonious reconstructions on a tree: A generalization of the Farris-Swofford-Maddison method. Discrete Applied Mathematics 56(2-3):245-265.

Narushima, H., and M. Hanazawa. 1997. A more efficient algorithm for MPR problems in phylogeny. Discrete Applied Mathematics 80(2-3):231-238.

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

Swofford, D. L., and W. P. Maddison. 1987. Reconstructing ancestral character states under Wagner parsimony. Mathematical Biosciences 87(2):199-229.

See Also

The functions described here are effectively wrapers of `phangorn`'s function `ancestral.pars`.

Examples

```
# let's write a quick & dirty ancestral trait plotting function

quickAncPlotter<-function(tree,ancData,cex){
ancCol<-(1:ncol(ancData))+1
plot(tree,show.tip.label=FALSE,no.margin=TRUE,direction="upwards")
tiplabels(pch=16,pie=ancData[(1:Ntip(tree)),],cex=cex,piecol=ancCol,
col=0)
nodelabels(pie=ancData[-(1:Ntip(tree)),],cex=cex,piecol=ancCol)
}

# example with retiolitid graptolite data

data(retiolitinae)
```

```
#unordered, MPR
ancMPR<-ancPropStateMat(retioTree, trait=retioChar[,2], type="MPR")
#unordered, ACCTRAN
ancACCTRAN<-ancPropStateMat(retioTree, trait=retioChar[,2], type="ACCTRAN")

#let's compare MPR versus ACCTRAN results
layout(1:2)
quickAncPlotter(retioTree,ancMPR,cex=0.5)
text(x=4,y=5,"type='MPR'",cex=1.5)
quickAncPlotter(retioTree,ancACCTRAN,cex=0.5)
text(x=5,y=5,"type='ACCTRAN'",cex=1.5)

minCharChange(retioTree,trait=retioChar[,2],type="MPR")
minCharChange(retioTree,trait=retioChar[,2],type="ACCTRAN")

# with simulated data

set.seed(444)
tree<-rtree(50)
#simulate under a likelihood model
char<-rTraitDisc(tree,k=3,rate=0.7)
tree$edge.length<-NULL
tree<-ladderize(tree)

#unordered, MPR
ancMPR<-ancPropStateMat(tree, trait=char, type="MPR")
#unordered, ACCTRAN
ancACCTRAN<-ancPropStateMat(tree, trait=char, type="ACCTRAN")
#ordered, MPR
ancMPRord<-ancPropStateMat(tree, trait=char, orderedChar=TRUE, type="MPR")

#let's compare MPR versus ACCTRAN results
layout(1:2)
quickAncPlotter(tree,ancMPR,cex=0.3)
text(x=8,y=15,"type='MPR'",cex=1.5)
quickAncPlotter(tree,ancACCTRAN,cex=0.3)
text(x=9,y=15,"type='ACCTRAN'",cex=1.5)
#MPR has much more uncertainty in node estimates
#but that doesn't mean ACCTRAN is preferable

#let's compare unordered versus ordered under MPR
layout(1:2)
quickAncPlotter(tree,ancMPR,cex=0.3)
text(x=8,y=15,"unordered char\nMPR",cex=1.5)
quickAncPlotter(tree,ancMPRord,cex=0.3)
text(x=9,y=15,"ordered char\nMPR",cex=1.5)
layout(1)

## Not run:
# what ancPropStateMat automates (with lots of checks):

require(phangorn)
```

```
char1<-matrix(char,,1)
rownames(char1)<-names(char)
#translate into something for phangorn to read
char1<-phyDat(char1,type="USER",levels=sort(unique(char1)))
x<-ancestral.pars(tree,char1,type="MPR")
y<-ancestral.pars(tree,char1,type="ACCTRAN")

## End(Not run)

#estimating minimum number of transitions with MPR
minCharChange(tree,trait=char,type="MPR")

#and now with ACCTRAN
minCharChange(tree,trait=char,type="ACCTRAN")

#POLYMORPHISM IN CHARACTER DATA


# example trait data with a polymorphic taxon
     # separated with '&' symbol
# similar to polymorphic data output by ReadMorphNexus from package Claddis
charPoly<-as.character(c(1,2,NA,0,0,1,"1&2",2,0,NA,0,2,1,1,"1&2"))
#simulate a tree with 16 taxa
set.seed(444)
tree<-rtree(15)
tree$edge.length<-NULL
tree<-ladderize(tree)
names(charPoly)<-tree$tip.label
charPoly

# need a contrast matrix that takes this into account
    #can build row by row, by hand

#first, build contrast matrix for basic states
contrast012<-rbind(c(1,0,0),c(0,1,0),c(0,0,1))
colnames(contrast012)<-rownames(contrast012)<-0:2
contrast012

#add polymorphic state and NA ambiguity as new rows
contrastPoly<-c(0,1,1)
contrastNA<-c(1,1,1)
contrastNew<-rbind(contrast012,'1&2'=contrastPoly,contrastNA)
rownames(contrastNew)[5]<-NA

#let's look at contrast
contrastNew

# now try this contrast table we've assembled
    # default: unordered, MPR
ancPoly<-ancPropStateMat(tree, trait=charPoly, contrast=contrastNew)

# but...!
# we can also do it automatically,
    # by default, states with '&' are automatically treated
    # as polymorphic character codings by ancPropStateMat
ancPolyAuto<-ancPropStateMat(tree, trait=charPoly, polySymbol="&")

# but does this match what the table we constructed?
ancPropStateMat(tree, trait=charPoly,
polySymbol="&", returnContrast=TRUE)

# compare to contrastNew above!
# only difference should be the default ambiguous
# character '?' is added to the table

#compare reconstructions
layout(1:2)
quickAncPlotter(tree,ancPoly,cex=0.5)
text(x=3.5,y=1.2,"manually-constructed\ncontrast",cex=1.3)
quickAncPlotter(tree,ancPolyAuto,cex=0.5)
text(x=3.5,y=1.2,"auto-constructed\ncontrast",cex=1.3)
layout(1)

#look pretty similar!

#i.e. the default polySymbol="&", but could be a different symbol
     #such as "," or "\"... it can only be *one* symbol, though

# all of this machinery should function just fine in minCharChange
# again, by default polySymbol="&" (included anyway here for kicks)
minCharChange(tree, trait=charPoly, polySymbol="&")
```

`modelMethods` *Model Function Methods: Parameter Names, Bounds and Initial Values*

Description

A large number of functions for obtaining and modifying the parameters of likelihood models made in paleotree. These functions allow users to obtain or set parameter names, or obtain and set parameter bounds, both of which are treated as an attribute of the function class used by paleotree. In practice, this allows users to quickly obtain parameter names and upper and lower values for use in bounded optimizers, including reasonable starting values.

Usage

```
parnames(x, ...)

## S3 method for class 'paleotreeFunc'
parnames(x, ...)
```

```
## S3 method for class 'constrained'
parnames(x, ...)

parnames(x) <- value

## S3 replacement method for class 'constrained'
parnames(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parnames(x) <- value

parbounds(x, ...)

## S3 method for class 'paleotreeFunc'
parbounds(x, ...)

## S3 method for class 'constrained'
parbounds(x, ...)

parbounds(x) <- value

## S3 replacement method for class 'constrained'
parbounds(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parbounds(x) <- value

parLower(x, ...)

## S3 method for class 'constrained'
parLower(x, ...)

## S3 method for class 'paleotreeFunc'
parLower(x, ...)

parLower(x) <- value

## S3 replacement method for class 'constrained'
parLower(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parLower(x) <- value

parUpper(x, ...)

## S3 method for class 'constrained'
parUpper(x, ...)
```

```
## S3 method for class 'paleotreeFunc'
parUpper(x, ...)

parUpper(x) <- value

## S3 replacement method for class 'constrained'
parUpper(x) <- value

## S3 replacement method for class 'paleotreeFunc'
parUpper(x) <- value

parInit(x, ...)

## S3 method for class 'constrained'
parInit(x, ...)

## S3 method for class 'paleotreeFunc'
parInit(x, ...)
```

Arguments

`x`	A function of S3 class 'paleotreeFunc' with all necessary attributes expected of that class, which include parameter names and upper and lower bounds. As I have deliberately not exported the function which creates this class, it should be impossible for regular users to obtain such objects easily without using one of the 'make' functions, which automatically output a function of the appropriate class and attributes.
`...`	'Ignored arguments to future methods' (i.e. for diversitree). Kept here only so constrainParPaleo is kept as close to the parent method in diversitree as possible.
`value`	The new value with which to replace the parameter names or bounds. Must be a vector of the same length as the number of parameters. For parbounds, must be a list composed of two vectors.

Details

Parameter names cannot be changed for a constrained function.

The parInit function calls the bounds for each parameter and gives a randomly selected value selected from a uniform distribution, using the parameter bounds for each parameter as the bounds on the uniform distribution. This users a shorthand to quickly generate initial parameter values which are within the set bounds, for use in functions such as `optim`. The random sampling of initial values allows a user to quickly assess if initial parameter values affect the optimization by simply rerunning the function on new values. Infinite initial parameter values (resulting from infinite bounds) are discarded, and replaced with the lower bound value (assuming only upper bounds are infinite...). Some randomly selected initial parameter values may be too high (due to the liberal upper bounds I set for parameters in many of the likelihood functions) and thus users should always try slightly different values to see if the resulting maximum likelihood parameter values change.

As parInit depends on the upper and lower bounds attribute, no function is offered to allow it to be replaced (as there is nothing to replace!).

Value

Returns the sought parameter names, bounds or initial values or (for the replacement methods) returns a modified function with the respective attributes altered.

Author(s)

These functions are strongly based on or inspired by the `argnames` functions provided for handling models in Rich Fitzjohn's library `diversitree`, but the functions presented here are derviations written by David Bapst.

See Also

These model methods were introduced to interact with the new model framework introduced in paleotree v1.9, in particular to interface with `constrainParPaleo`.

Examples

```
#example with make_durationFreqCont
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa,r=0.5)
likFun <- make_durationFreqCont(rangesCont)

#get parameter names
parnames(likFun)

#get the bounds for those parameters
parbounds(likFun)

#can also get these seperately
parLower(likFun)
parUpper(likFun)

#initial parameter values
parInit(likFun)   #arbitrary midway value between par bounds

#can then use these in optimizers, such as optim with L-BFGS-B
#see the example for make_durationFreqCont

#renaming parameter names
likFun2 <- likFun
parnames(likFun2) <- c("extRate","sampRate")
parnames(likFun2)
#test if reset correctly
parnames(likFun2)==c("extRate","sampRate")
#also works for constrained functions
constrainFun<-constrainParPaleo(likFun,q.1~r.1)
parnames(constrainFun)
#also modified the parameter bounds, see!
```

```
parbounds(constrainFun)
parInit(constrainFun)
#but cannot rename parameter for constrained function!
```

---

modifyTerminalBranches *Modify, Drop or Bind Terminal Branches of Various Types (Mainly for Paleontological Phylogenies)*

Description

These functions modify terminal branches or drop certain terminal branches based on various criteria. DropZLB drops tip-taxa that are attached to the tree via zero-length terminal branches ("ZLBs"). This is sometimes useful for paleo-trees, as various time-scaling methods often produce these ZLBs, taxa whose early appearance causes them to be functionally interpreted as ancestors in some time-scaling methods. Removing ZLBs is advised for analyses of diversification/diversity, as these will appear as simultaneous speciation/extinction events. Note this function only drops tips attached to a terminal zero-length branch; if you want to collapse internal zero-length branches, see the ape function `di2multi`.

Usage

```
dropZLB(tree)

dropExtinct(tree, tol = 0.01, ignore.root.time = FALSE)

dropExtant(tree, tol = 0.01)

addTermBranchLength(tree, addtime = 0.001)

fixRootTime(treeOrig, treeNew, consistentDepth = TRUE,
  nodeAgeTransfer = TRUE)

dropPaleoTip(tree, ...)

bindPaleoTip(tree, tipLabel, nodeAttach = NULL, tipAge = NULL,
  edgeLength = NULL, positionBelow = 0, noNegativeEdgeLength = TRUE)
```

Arguments

`tree`	A phylogeny as a phylo object. dropPaleoTip requires this input object to also have a $root.time element. If not provided for bindPaleoTip, then the root.time will be presumed to be such that the furthest tip from the root is at time=0.
`tol`	Tolerance for determining modern age; used for distinguishing extinct from extant taxa. Tips which end within 'tol' of the furthest distance from the root will be treated as 'extant' taxa for the purpose of keeping or dropping.

`ignore.root.time`	Ignore root.time in calculating which tips are extinct? root.time will still be adjusted
`addtime`	Extra amount of time to add to all terminal branch lengths.
`treeOrig`	A phylo object of a time-scaled phylogeny with a $root.time element
`treeNew`	A phylo object containing a modified form of treeOrig (with no extra tip taxa added, but possibly with some tip taxa removed).
`consistentDepth`	A logical, either TRUE or FALSE. If TRUE (the default) the tree's root-to-furthest-tip depth is tested to make sure this depth is not greater than the new root.time appended to the output tree.
`nodeAgeTransfer`	A logical. If TRUE, the root.time of the new tree is determined by comparing the clades of taxa between the two input trees. The new root age assigned is the age of (*1*) the treeOrig clade that contains *all* taxa present in treeNew and, if the set of (1) contains multiple clades, (*2*) the clade in the (1) set that contains the fewest taxa not in treeNew. If FALSE, the root.time assigned to treeNew is the root.time of treeOrig, adjusted based on the change in total tree depth between treeOrig and treeNew, as measured between the root and the first matching taxon in both trees. The later is how fixRootTime functioned by default prior to paleotree v2.3.
`...`	additional arguments passed to dropPaleoTip are passed to drop.tip
`tipLabel`	A character string of `length = 1` containing the name of the new tip to be added to `tree`.
`nodeAttach`	Node or tip ID number (as given in tree$edge) at which to attach the new tip. See documentation of `bind.tip` for more details.
`tipAge`	The age of the tip taxon added to the tree, in time before present (i.e. where present is 0), given in the same units as the edges of the tree are already scaled. Cannot be given if `edgeLength` is given.
`edgeLength`	The new `edge.length` of the terminal branch this tip is connected to. Cannot be given if `tipAge` is given.
`positionBelow`	The distance along the edge below the node to be attached to (given in codenodeAttach to add the new tip. Cannot be negative or greater than the length of the edge below `nodeAttach`.
`noNegativeEdgeLength`	Return an error if a negative terminal edge length is calculated for the new tip.

Details

DropExtinct drops all terminal branches which end before the modern (i.e. extinct taxa). DropExtant drops all terminal branches which end at the modern (i.e. extant/still-living taxa). In both cases, the modern is defined based on tree$root.time if available, or the modern is inferred to be the point in time when the tip furthest from the root (the latest tip) terminates.

If the input tree has a $root.time element, as expected for most paleo-tree objects handled by this library, that root.time is adjusted if the relative time of the root divergence changes when terminal branches are dropped. Adjusted root.times are only given if the input tree has root.times.

addTermBranchLength adds an amount equal to the argument 'addtime' to the terminal branch lengths of the tree. If there is a $root.time element, this is increased by an amount equal to addtime. A negative amount can be input to reduce the length of terminal branches. However, if negative branch lengths are produced, the function fails and a warning is produced. This function does *not* call fixRootTime, so the root.time elements in the result tree may be nonsensical, particularly if negative amounts are input.

When a tree is modified, such as having tips dropped or branches extended, fixRootTime can be used to find the new $root.time. It is mainly used as a utility function called by a majority of the other functions discussed in this help file.

dropPaleoTip is a wrapper for ape's `drop.tip` which also modifies the $root.time element if necessary, using `fixRootTime`. Similarly, bindPaleoTip is a wrapper for phytool's `bind.tip` which allows tip age as input and modifies the $root.time element if necessary (i.e. if a tip is added to edge leading up to the root).

Note that for bindPaleoTip, tips added below the root are subtracted from any existing $root.edge element, as per behavior of bind.tip and bind.tree. However, bindPaleoTip will append a $root.edge of the appropriate length if one does not exist (or is not long enough) to avoid an error. After binding is finished, any $root.edge equal to 0 is removed before the resulting tree is output.

Value

Gives back a modified phylogeny as a phylo object, generally with a modified $root.time element.

Author(s)

David W. Bapst. The functions dropTipPaleo and bindTipPaleo are modified imports of `drop.tip` and `bind.tip` from packages `ape` and `phytools`.

See Also

`compareTermBranches`, `phyloDiv`, `drop.tip`, `bind.tip`

Examples

```
set.seed(444)
#Simulate some fossil ranges with simFossilRecord
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont<-sampleRanges(taxa,r=0.5)
#Now let's make a tree using taxa2phylo
tree <- taxa2phylo(taxa,obs_time=rangesCont[,2])
#compare the two trees
layout(1:2)
plot(ladderize(tree))
plot(ladderize(dropZLB(tree)))
layout(1)

#example using dropExtinct and dropExtant
```

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=c(10,20))
taxa<-fossilRecord2fossilTaxa(record)
tree<-taxa2phylo(taxa)
phyloDiv(tree)
tree1 <- dropExtinct(tree)
phyloDiv(tree1)
tree2 <- dropExtant(tree)
phyloDiv(tree2)

#graphics.off()

#example using addTermBranchLength
set.seed(444)
treeA <- rtree(10)
treeB <- addTermBranchLength(treeA,1)
compareTermBranches(treeA,treeB)

#########################
#test dropPaleoTip
#(and fixRootTime by extension...)

#simple example
tree<-read.tree(text="(A:3,(B:2,(C:5,D:3):2):3);")
tree$root.time<-10
plot(tree,no.margin=FALSE)
axisPhylo()

# now a series of tests, dropping various tips
(test<-dropPaleoTip(tree,"A")$root.time) # =7
(test[2]<-dropPaleoTip(tree,"B")$root.time) # =10
(test[3]<-dropPaleoTip(tree,"C")$root.time) # =10
(test[4]<-dropPaleoTip(tree,"D")$root.time) # =10
(test[5]<-dropPaleoTip(tree,c("A","B"))$root.time) # =5
(test[6]<-dropPaleoTip(tree,c("B","C"))$root.time) # =10
(test[7]<-dropPaleoTip(tree,c("A","C"))$root.time) # =7
(test[8]<-dropPaleoTip(tree,c("A","D"))$root.time) # =7

# is it all good? if not, fail so paleotree fails...
if(!identical(test,c(7,10,10,10,5,10,7,7))){stop("fixRootTime fails!")}


##############
#testing bindPaleoTip

# simple example
tree<-read.tree(text="(A:3,(B:2,(C:5,D:3):2):3);")
tree$root.time<-20
plot(tree,no.margin=FALSE)
axisPhylo()

## Not run:
```

```
require(phytools)

#bindPaleoTip effectively wraps bind.tip from phytools
# using a conversion like below

tipAge<-5
node<-6

#new length = the root time - tipAge - nodeheight(tree,node)

newLength<-tree$root.time-tipAge-nodeheight(tree,node)
tree1<-bind.tip(tree,"tip.label",where=node,edge.length=newLength)

layout(1:2)
plot(tree);axisPhylo()
plot(tree1);axisPhylo()


## End(Not run)

# now with bindPaleoTip


tree1<-bindPaleoTip(tree,"new",nodeAttach=6,tipAge=5)

layout(1:2)
plot(tree);axisPhylo()
plot(tree1);axisPhylo()
layout(1)

#then the tip age of "new" should 5
test<-dateNodes(tree1)[which(tree1$tip.label=="new")]==5
if(!test){stop("bindPaleoTip fails!")}

# with positionBelow

tree1<-bindPaleoTip(tree,"new",nodeAttach=6,tipAge=5,positionBelow=1)

layout(1:2)
plot(tree);axisPhylo()
plot(tree1);axisPhylo()
layout(1)

# at the root

tree1<-bindPaleoTip(tree,"new",nodeAttach=5,tipAge=5)

layout(1:2)
plot(tree);axisPhylo()
plot(tree1);axisPhylo()
layout(1)
```

```
#then the tip age of "new" should 5
test<-dateNodes(tree1)[which(tree1$tip.label=="new")]==5
if(!test){stop("bindPaleoTip fails!")}

# at the root with positionBelow

tree1<-bindPaleoTip(tree,"new",nodeAttach=5,tipAge=5,
positionBelow=3)

layout(1:2)
plot(tree);axisPhylo()
plot(tree1);axisPhylo()
layout(1)

#then the tip age of "new" should 5
test<-dateNodes(tree1)[which(tree1$tip.label=="new")]==5
#and the root age should be 23
test1<-tree1$root.time==23
if(!test | !test1){stop("bindPaleoTip fails!")}
```

`multiDiv` *Calculating Diversity Curves Across Multiple Datasets*

Description

Calculates multiple diversity curves from a list of datasets of taxic ranges and/or phylogenetic trees, for the same intervals, for all the individual datasets. A median curve with 95 percent quantile bounds is also calculated and plotted for each interval.

Usage

```
multiDiv(data, int.length = 1, plot = TRUE, split.int = TRUE,
  drop.ZLB = TRUE, drop.cryptic = FALSE, extant.adjust = 0.01,
  plotLogRich = FALSE, timelims = NULL, int.times = NULL,
  plotMultCurves = FALSE, multRainbow = TRUE, divPalette = NULL,
  divLineType = 1, main = NULL)

plotMultiDiv(results, plotLogRich = FALSE, timelims = NULL,
  plotMultCurves = FALSE, multRainbow = TRUE, divPalette = NULL,
  divLineType = 1, main = NULL)
```

Arguments

`data`	A list where each element is a dataset, formatted to be input in one of the diversity curve functions listed in `DiversityCurves`.
`int.length`	The length of intervals used to make the diversity curve. Ignored if int.times is given.

`plot`	If `TRUE`, the median diversity curve is plotted.
`split.int`	For discrete time data, should calculated/input intervals be split at discrete time interval boundaries? If FALSE, can create apparent artifacts in calculating the diversity curve. See below.
`drop.ZLB`	If true, zero-length terminal branches are dropped from the input tree for phylogenetic datasets, before calculating standing diversity.
`drop.cryptic`	If TRUE, cryptic taxa are merged to form one taxon for estimating taxon curves. Only works for objects from `simFossilRecord` via `fossilRecord2fossilTaxa`.
`extant.adjust`	Amount of time to be added to extend start time for (0,0) bins for extant taxa, so that the that 'time interval' doesn't appear to have an infinitely small width.
`plotLogRich`	If TRUE, taxic diversity is plotted on log scale.
`timelims`	Limits for the x (time) axis for diversity curve plots. Only affects plotting. Given as either NULL (the default) or as a vector of length two as for 'xlim' in the basic R function plot. Time axes will be plotted *exactly* to these values.
`int.times`	An optional two-column matrix of the interval start and end times for calculating the diversity curve. If NULL, calculated internally. If given, the argument split.int and int.length are ignored.
`plotMultCurves`	If `TRUE`, each individual diversity curve is plotted rather than the median diversity curve and 95 percent quantiles. FALSE by default.
`multRainbow`	If `TRUE` and plotMultCurves are both `TRUE`, each line is plotted as a different, randomized color using the function 'rainbow'. If FALSE, each line is plotted as a black line. This argument is ignored if divPalette is supplied.
`divPalette`	Can be used so users can pass a vector of chosen color identifiers for each diversity curve in 'data' which will take precedence over multRainbow. Must be the same length as the number of diversity curves supplied.
`divLineType`	Used to determine line type (`lty`) of the diversity curves plotted when `plotMultCurves = TRUE`. Default is `lty = 1` for all curves. Must be either length of 1 or exact length as number of diversity curves.
`main`	The main label for the figure.
`results`	The output of a previous run of multiDiv for replotting.

Details

This function is essentially a wrapper for the individual diversity curve functions included in paleotree. multiDiv will intuitively decide whether input datasets are continuous-time taxic ranges, discrete-time (binned interval) taxic ranges or phylogenetic trees, as long as they are formatted as required by the respective diversity curve functions. A list that contains a mix of data types is entirely acceptable. A list of matrices output from `fossilRecord2fossilTaxa`, via simulation with `simFossilRecord` is allowable, and treated as input for `taxicDivCont`. Data of an unknown type gives back an error.

The argument split.int splits intervals, if and only if discrete interval time data is included among the datasets. See the help file for taxicDivDisc to see an explanation of why `split.int = TRUE` by default is probably a good thing.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

The 'averaged' curve is actually the median rather than the mean as diversity counts are often highly skewed (in this author's experience).

The shaded certainty region around the median curve is the two-tailed 95 percent lower and upper quantiles, calculated from the observed data. It is not a true probabilisitic confidence interval, as it has no relationship to the standard error.

Value

A list composed of three elements will be invisibly returned:

`int.times`	A two column matrix giving interval start and end times
`div`	A matrix of measured diversities in particular intervals by rows, with each column representing a different dataset included in the input
`median.curve`	A three column matrix, where the first column is the calculated median curve and the second and third columns are the 95 percent quantile upper and lower bounds

See Also

The diversity curve functions used include: `phyloDiv`, `taxicDivCont` and `taxicDivDisc`.

Also see the function LTT.average.root in the package TreeSim, which calculates an average LTT curve for multiple phylogenies, the functions mltt.plot in ape and ltt in phytools.

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
rangesCont <- sampleRanges(taxa, r=0.5)
rangesDisc <- binTimeData(rangesCont, int.length=1)
cladogram<-taxa2cladogram(taxa, plot=TRUE)
#using multiDiv with very different data types
ttree <- timePaleoPhy(cladogram, rangesCont, type="basic", add.term=TRUE, plot=FALSE)
input <- list(rangesCont, rangesDisc, ttree)
multiDiv(input, plot=TRUE)

#using fixed interval times
multiDiv(input, int.times=rangesDisc[[1]], plot=TRUE)

#using multiDiv with samples of trees
ttrees <- timePaleoPhy(cladogram, rangesCont, type="basic",
    randres=TRUE, ntrees=10, add.term=TRUE)
multiDiv(ttrees)
#uncertainty in diversity history is solely due to
   #the random resolution of polytomies

#multiDiv can also take output from simFossilRecord, via fossilRecord2fossilTaxa
#what do many simulations run under some set of conditions 'look' like on average?
```

```
set.seed(444)
records<-simFossilRecord(p=0.1, q=0.1, nruns=10,
 totalTime=30, plot=TRUE)
taxa<-sapply(records,fossilRecord2fossilTaxa)
multiDiv(taxa)
#increasing cone of diversity!
#Even better on a log scale:
multiDiv(taxa, plotLogRich=TRUE)

#pure-birth example with simFossilRecord
#note that conditioning is tricky
set.seed(444)
recordsPB<-simFossilRecord(p=0.1, q=0, nruns=10,
 totalTime=30,plot=TRUE)
taxaPB<-sapply(recordsPB,fossilRecord2fossilTaxa)
multiDiv(taxaPB,plotLogRich=TRUE)

#compare many discrete diversity curves
discreteRanges<-lapply(taxa,function(x)
binTimeData(sampleRanges(x, r=0.5,
     min.taxa=1), int.length=7))
multiDiv(discreteRanges)

layout(1)
```

`nearestNeighborDist` *Nearest Neighbor Distances for Morphological Disparity Studies*

Description

This is a simple function for obtaining nearest neighbor distance from a symmetric pair-wises distance matrix, assumed here to be dissimilarities between pairs of taxa. Per-species NND is returned rather than a mean or other summary value.

Usage

```
nearestNeighborDist(distMat)
```

Arguments

`distMat`	A symmetric, square pair-wise distance matrix, assumed to be a dissimilarity matrix with a zero diagonal. Can be a 'dist' object rather than a numerical matrix. Taxon labels can be applied to the rows and columns (or as labels if a 'dist' object) and will be used to name the resulting output.

Details

This function is mainly included here for pedagogical (teaching) purposes. NND is so simple to calculate, users are urged to write their own functions for primary research purposes.

Typically, the *mean* NND for a group is reported and used to compare different groupings of taxa (such as different time intervals, or different clades). Bootstrapping should be used to generate confidence intervals.

Value

Returns a vector of the nearest neighbor distance for each unit (taxon) in the pair-wise distance matrix, named with the labels from the input distance matrix.

Author(s)

David W. Bapst

References

Bapst, D. W., P. C. Bullock, M. J. Melchin, H. D. Sheets, and C. E. Mitchell. 2012. Graptoloid diversity and disparity became decoupled during the Ordovician mass extinction. *Proceedings of the National Academy of Sciences* 109(9):3428-3433.

Ciampaglio, C. N., M. Kemp, and D. W. McShea. 2001. Detecting changes in morphospace occupation patterns in the fossil record: characterization and analysis of measures of disparity. *Paleobiology* 27(4):695-715.

Foote, M. 1990. Nearest-neighbor analysis of trilobite morphospace. *Systematic Zoology* 39:371-382.

See Also

For the example dataset used in examples, see `graptDisparity`

Examples

```
#example using graptolite disparity data from Bapst et al. 2012

#load data
data(graptDisparity)

#calculate mean NND
NND<-nearestNeighborDist(graptDistMat)
mean(NND)

#calculate NND for different groups

#group (clade/paraclade) coding
groupID <- graptCharMatrix[,54]+1

groupNND<-numeric(7)
names(groupNND)<-c("Normalo.","Monogr.","Climaco.",
   "Dicrano.","Lasiogr.","Diplogr.","Retiol.")
for(i in unique(groupID)){
   groupNND[i]<-mean(nearestNeighborDist(
      graptDistMat[groupID==i,groupID==i]))
   }
groupNND

#the paraphyletic Normalograptids that survived the HME are most clustered
   #but this looks at all the species at once
   #and doesn't look for the nearest *co-extant* neighbor!
   #need to bring in temporal info to test that
```

`occData2timeList` *Converting Occurrences Data to a timeList Data Object*

Description

This function converts occurrence data, given as a list where each element is a different taxon's occurrence table (containing minimum and maximum ages for each occurrence), to the 'timeList' format, consisting of a list composed of a matrix of lower and upper age bounds for intervals, and a second matrix recording the interval in which taxa first and last occur in the given dataset.

Usage

```
occData2timeList(occList, intervalType = "dateRange")
```

Arguments

`occList`	A list where every element is a table of occurrence data for a different taxon, such as that returned by `taxonSortPBDBocc`. The occurrence data can be either a two-column matrix composed of the lower and upper age bounds on each taxon occurrence, or has two named variables which match any of the field names given by the PBDB API under either the 'pbdb' vocab or 'com' (compact) vocab for early and late age bounds.
`intervalType`	Must be either "dateRange" (the default), "occRange" or "zoneOverlap". Please see details below.

Details

This function should translate taxon-sorted occurrence data, which could be Paleobiology Database datasets sorted by `taxonSortPBDBocc` or any data object where occurrence data (i.e. age bounds for each occurrence) for different taxa is separated into different elements of a named list.

The argument intervalType:

The argument `intervalType` controls the algorithm used for obtain first and last interval bounds for each taxon, of which there are several to select from:intervalType

"dateRange" The default option. The bounds on the first appearances are the span between the oldest upper and lower bounds of the occurrences, and the bounds on the last appearances are the span between the youngest upper and lower bounds across all occurrences. This is guaranteed to provide the smallest bounds on the first and last appearances, and was originally suggested to the author by J. Marcot.

"occRange" This option returns the smallest bounds among (a) the oldest occurrences for the first appearance (i.e. all occurrences with their lowest bound at the oldest lower age bound), and (b) the youngest occurrences for the last appearance (i.e. all occurrences with their uppermost bound at the youngest upper age bound).

"zoneOverlap" This option is an attempt to mimic the stratigraphic range algorithm used by PBDB Classic which "finds the oldest base that is older than at least part of all the intervals and the youngest that is younger than at least part of all the intervals" (pers.comm., J. Alroy). This is a somewhat more complex case as we are trying to obtain a `timeList` object. So, for calculating the bounds of the first interval a taxon occurs in, the `zoneOverlap` algorithm looks for all occurrences that overlap with the age range of the earliest-most occurrence and (1) obtains their earliest boundary ages and returns the latest-most earliest age boundary among these overlapping occurrences and (2) obtains their latest boundary ages and returns the earliest-most latest age boundary among these overlapping occurrences. Similarly, for calculating the bound of the last interval a taxon occurs in, the `zoneOverlap` algorithm looks for all occurrences that overlap with the age range of the latest-most occurrence and (1) obtains their earliest boundary ages and returns the latest-most earliest age boundary among these overlapping occurrences and (2) obtains their latest boundary ages and returns the earliest-most latest age boundary among these overlapping occurrences.
On theoretical grounds, one could probably describe the zone-of-overlap algorithm as minimizing taxonomic age ranges by assuming that all overlapping occurrences at the start and end of a taxon's range probably describe a very similar first and last appearance (FADs and LADs), and thus picks the occurrence with bounds that extends the taxonomic range the least. However, this does come with a downside that if these occurrences are not essentially repeated attempts to capture the same FAD or LAD, then the zone-of-overlap algorithm isn't an accurate depiction of the uncertainty in the ages. The true biological range of a taxon might be well outside the bounds obtained using the zone-of-overlap algorithm. A more conservative approach is the `"dateRange"` algorithm which finds the smallest possible bounds on the endpoints of a taxon's range without ignoring uncertainty from any particular set of occurrences.

Value

Returns a standard timeList data object, as used by many other paleotree functions, like `bin_timePaleoPhy`, `bin_cal3TimePaleoPhy` and `taxicDivDisc`

Author(s)

David W. Bapst, with the 'dateRange' algorithm suggested by Jon Marcot.

See Also

`taxonSortPBDBocc`, `plotOccData` and the example graptolite dataset at `graptPBDB`

Examples

```
data(graptPBDB)

graptOccSpecies<-taxonSortPBDBocc(graptOccPBDB,rank="species",onlyFormal=FALSE)
graptTimeSpecies<-occData2timeList(occList=graptOccSpecies)

head(graptTimeSpecies[[1]])
head(graptTimeSpecies[[2]])

graptOccGenus<-taxonSortPBDBocc(graptOccPBDB,rank="genus",onlyFormal=FALSE)
graptTimeGenus<-occData2timeList(occList=graptOccGenus)

layout(1:2)
taxicDivDisc(graptTimeSpecies)
taxicDivDisc(graptTimeGenus)

# the default interval calculation is "dateRange"
# let's compare to the other option, "occRange"
# for species

graptOccRange<-occData2timeList(occList=graptOccSpecies, intervalType="occRange")

#we would expect no change in the diversity curve
#because there are only changes in th
#earliest bound for the FAD
#latest bound for the LAD
#so if we are depicting ranges within maximal bounds
#dateRanges has no effect
layout(1:2)
taxicDivDisc(graptTimeSpecies)
taxicDivDisc(graptOccRange)
#yep, identical

#so how much uncertainty was gained by using dateRange?

# write a simple function for getting uncertainty in first and last
# appearance dates from a timeList object
sumAgeUncert<-function(timeList){
fourDate<-timeList2fourDate(timeList)
perOcc<-(fourDate[,1]-fourDate[,2])+(fourDate[,3]-fourDate[,4])
sum(perOcc)
}

#total amount of uncertainty in occRange dataset
sumAgeUncert(graptOccRange)
#total amount of uncertainty in dateRange dataset
sumAgeUncert(graptTimeSpecies)
#the difference
sumAgeUncert(graptOccRange)-sumAgeUncert(graptTimeSpecies)
#as a proportion
1-(sumAgeUncert(graptTimeSpecies)/sumAgeUncert(graptOccRange))
```

```
#a different way of doing it
dateChange<-timeList2fourDate(graptTimeSpecies)-timeList2fourDate(graptOccRange)
apply(dateChange,2,sum)
#total amount of uncertainty removed by dateRange algorithm
sum(abs(dateChange))

layout(1)
```

`optimPaleo` *Simplified Optimizer for paleotree Likelihood Functions*

Description

This function is a deliberately simplistic automater for the function `optim` and the use of the "L-BFGS-B" optimizing method, with initial parameter values and bounds provided with `parInit`, `parLower` and `parUpper`. It is mainly provided here as a shorthand to be used in educational demonstrations where model-fitting is not the primary focus, and use in actual analyses should be avoided.

Usage

```
optimPaleo(modelFun)
```

Arguments

`modelFun`	A likelihood function for a model, of class 'paleotreeFunc'.

Details

This is mainly provided in this publicly released package for pedagogical reasons. Users seeking an optimizer for their own analytical purposes should write their own optim function.

Value

Returns the results from using optim.

See Also

`constrainParPaleo` and `modelMethods`

Examples

```

# This function simply replicates optim() as shown below
    # where modelFun is the likelihood function

#optim(parInit(modelFun),modelFun,
# lower=parLower(modelFun),upper=parUpper(modelFun),
# method="L-BFGS-B",control=list(maxit=1000000))
```

parentChild2taxonTree *Create a Taxonomy-Based Phylogeny ('Taxon Tree') from a Table of Parent-Child Taxon Relationships*

Description

This function takes a two-column matrix of taxon names, indicating a set of binary parent-taxon:child-taxon paired relationships with a common root, and returns a 'taxonomy-tree' phylogeny object of class 'phylo'.

Usage

```
parentChild2taxonTree(parentChild, tipSet = "nonParents", cleanTree = TRUE)
```

Arguments

`parentChild`	A two-column matrix of type `character` where each element is a taxon name. Each row represents a parent-child relationship with first the parent (column 1) taxon name and then the child (column 2).
`tipSet`	This argument controls which taxa are selected as tip taxa for the output tree. The default `tipSet="nonParents"` selects all child taxa which are not listed as parents in `parentChild`. Alternatively, `tipSet="all"` will add a tip to every internal node with the parent-taxon name encapsulated in parentheses.
`cleanTree`	By default, the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If FALSE, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.

Details

All taxa listed must be traceble via their parent-child relationships to a single, common ancestor which will act as the root node for output phylogeny. Additionally, the root used will be the parent taxon to all tip taxa closest in terms of parent-child relationships to the tip taxa: i.e., the most recent common ancestor. Ancestral taxa which are singular internal nodes that trace to this root are removed, and a message is printed.

Value

A phylogeny of class 'phylo', with tip taxa as controlled by argument `tipSet`. The output tree is returned with no edge lengths.

The names of higher taxa than the tips should be appended as the element $node.label for the internal nodes.

Author(s)

David W. Bapst

See Also

makePBDBtaxonTree, taxonTable2taxonTree

Examples

```
#let's create a small, really cheesy example
pokexample<-rbind(cbind("Squirtadae",c("Squirtle","Blastoise","Wartortle")),
c("Shelloidea","Lapras"),c("Shelloidea","Squirtadae"),
c("Pokezooa","Shelloidea"),c("Pokezooa","Parasect"),
c("Rodentapokemorpha","Linoone"),c("Rodentapokemorpha","Sandshrew"),
c("Rodentapokemorpha","Pikachu"),c("Hirsutamona","Ursaring"),
c("Hirsutamona","Rodentapokemorpha"),c("Pokezooa","Hirsutamona"))

#Default: tipSet='nonParents'
pokeTree<-parentChild2taxonTree(pokexample, tipSet="nonParents")
plot(pokeTree);nodelabels(pokeTree$node.label)

#Get ALL taxa as tips with tipSet='all'
pokeTree<-parentChild2taxonTree(pokexample, tipSet="all")
plot(pokeTree);nodelabels(pokeTree$node.label)


## Not run:

# let's try a dataset where not all the taxon relationships lead to a common root

pokexample_bad<-rbind(cbind("Squirtadae",c("Squirtle","Blastoise","Wartortle")),
c("Shelloidea","Lapras"),c("Shelloidea","Squirtadae"),
c("Pokezooa","Shelloidea"),c("Pokezooa","Parasect"),
c("Rodentapokemorpha","Linoone"),c("Rodentapokemorpha","Sandshrew"),
c("Rodentapokemorpha","Pikachu"),c("Hirsutamona","Ursaring"),
c("Hirsutamona","Rodentapokemorpha"),c("Pokezooa","Hirsutamona"),
c("Umbrarcheota","Gengar"))

#this should return an error, as Gengar doesn't share common root
pokeTree<-parentChild2taxonTree(pokexample_bad)


# another example, where a taxon is listed as both parent and child
pokexample_bad2<-rbind(cbind("Squirtadae",c("Squirtle","Blastoise","Wartortle")),
c("Shelloidea",c("Lapras","Squirtadae","Shelloidea")),
c("Pokezooa","Shelloidea"),c("Pokezooa","Parasect"),
c("Rodentapokemorpha","Linoone"),c("Rodentapokemorpha","Sandshrew"),
c("Rodentapokemorpha","Pikachu"),c("Hirsutamona","Ursaring"),
c("Hirsutamona","Rodentapokemorpha"),c("Pokezooa","Hirsutamona"),
c("Umbrarcheota","Gengar"))

#this should return an error, as Shelloidea is its own parent
pokeTree<-parentChild2taxonTree(pokexample_bad2)
```

```
## End(Not run)



# note that we should even be able to do this with ancestor-descendent pairs from
 # simulated datasets from simFossilRecord, like so:
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
# need to reorder the columns so parents (ancestors) first, then children
parentChild2taxonTree(taxa[,2:1])
# now note that it issues a warning that the input wasn't type character
   # and it will be coerced to be such
```

perCapitaRates *perCapitaRates*

Description

Calculates and plots per-capita origination and extinction rates from sequential discrete-time taxon ranges, following Foote (2000).

Usage

```
perCapitaRates(timeList, plot = TRUE, logRates = FALSE,
  drop.extant = FALSE, isExtant = NULL, jitter = TRUE,
  legendPosition = "topleft")
```

Arguments

`timeList`	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. See details.
`plot`	If true, the per-capita origination and extinctions rates are plotted for each interval. Rates which cannot be calculated for an interval will not be plotted, thus appearing as a gap in the plotted graph. The author takes no responsibility for the aesthetics of this plot.
`logRates`	If true, rates plotted on log scale.
`drop.extant`	Drops all extant taxa from a dataset before calculating per-capita origination and extinction rates.
`isExtant`	A vector of TRUE and FALSE values, same length as the number of taxa in the second matrix of timeList, where TRUE values indicate taxa that are alive in the modern day (and thus are boundary crossers which leave the most recent interval). By default, this argument is NULL and instead which taxa are extant is inferred based on which taxa occur in an interval with start and end times both equal to zero. See details.

`jitter`	If TRUE (default) the extinction rate will be plotted slightly ahead of the origination rate on the time axis, so the two can be differentiated.
`legendPosition`	The position of a legend indicating which line is origination rate and which is extinction rate on the resulting plot. This is given as the possible positions for argument 'x' of the function `legend`, and by default is "topleft", which will be generally useful if origination and extinction rates are initially low. If legendPosition is NA, then a legend will not be plotted.

Details

This function calculates the per-capita rates of taxonomic origination and extinction from paleontological range data, as described by Foote (2000). These values are the instantaneous rate of either type of event occurring per lineage time-units. Although Foote (2001) also presents a number of alternative rates collected from the prior literature such as the VanValen rate metrics, these are not implemented here, but could be estimated using the matrix invisibly output by this function.

The timeList object should be a list composed of two matrices, the first matrix giving by-interval start and end times (in absolute time), the second matrix giving the by-taxon first and last appearances in the intervals defined in the first matrix, numbered as the rows. Absolute time should be decreasing, while the intervals should be numbered so that the number increases with time. Taxa alive in the modern should be either (a) listed in isExtant or (b) listed as last occurring in a time interval that begins at time 0 and ends at time 0. See the documentation for the time-scaling function `bin_timePaleoPhy` and the simulation function `binTimeData` for more information on formatting.

Unlike some functions in paleotree, such as the diversity curve functions, intervals must be both sequential and non-overlapping. The diversity curve functions deal with such issues by assuming taxa occur from the base of the interval they are first found in until the end of the last interval they are occur in. This inflation of boundary crossers could badly bias estimates of per-capita diversification rates.

Value

This function will invisibly return a ten column matrix, where the number of rows is equal to the number of intervals. The first two columns are interval start and end times and the third column is interval length. The fourth through eighth column is the four fundamental classes of taxa from Foote (2001): Nbt, NbL, NFt, NFL and their sum, N. The final two columns are the per-capita rates estimated for each interval in units per lineage time-units; the ninth column is the origination rate ('pRate') and the tenth column is the extinction rate ('qRate').

References

Foote, M. 2000 Origination and extinction components of taxonomic diversity: general problems. Pp. 74–102. In D. H. Erwin, and S. L. Wing, eds. Deep Time: Paleobiology's Perspective. The Paleontological Society, Lawrence, Kansas.

See Also

`DiversityCurves`, `SamplingConv`

Examples

```
#with the retiolinae dataset
data(retiolitinae)
perCapitaRates(retioRanges)

#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(80,100), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5)
#Now let's use binTimeData() to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length=5)
#and get the per-capita rates
perCapitaRates(rangesDisc)
#on a log scale
perCapitaRates(rangesDisc,logRates=TRUE)

#get mean and median per-capita rates
res<-perCapitaRates(rangesDisc,plot=FALSE)
apply(res[,c("pRate","qRate")],2,mean,na.rm=TRUE)
apply(res[,c("pRate","qRate")],2,median,na.rm=TRUE)

#with modern taxa
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(80,100))
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.5,,modern.samp.prob=1)
#Now let's use binTimeData() to bin in intervals of 5 time units
rangesDisc <- binTimeData(rangesCont,int.length=5)
#and now get per-capita rates
perCapitaRates(rangesDisc)
```

perfectParsCharTree *Simulate a Set of Parsimony-Informative Characters for a Phylogeny*

Description

Creates a simulated set of parsimony-informative characters for a given rooted phylogeny, with characters shared out equally across nodes in the phylogeny, with any remaining characters assigned randomly to nodes.

Usage

```
perfectParsCharTree(tree, nchar)
```

Arguments

`tree`	A phylogeny of class 'phylo'
`nchar`	Number of parsimonious binary characters to simulate on the phylogeny.

Details

This function takes some a tree and places a number of binary characters on the tree, with character states arranged as if the derived condition was gained once, at a single node, and never lost. This ensures that the resulting simulated character matrices have no character conflict, supporting a single solution under maximum parsimony.

If `nchar` is greater than the number of nodes on the input phylogeny (ignoring the root), then characters are first placed to evenly cover all nodes, with as many full passes of tree as possible. Any characters in excess are placed at random nodes, without replacement. In other words, if a tree has 10 nodes (plus the root) and 25 characters are simulated, 20 of those characters will consist of two 10-character 'full passes' of the tree. The remaining five will be randomly dropped on the tree.

If few characters are simulated than the number of nodes, these are randomly placed on the given topology without replacement, just as described above.

This function assumes, like almost every function in paleotree, that the tree given is rooted, even if the most basal node is a polytomy.

Value

A matrix of `nchar` parsimonious binary characters for each taxon on `tree`, with states 0 and 1.

Author(s)

David W. Bapst

Examples

```
data(retiolitinae)

#fewer characters than nodes
perfectParsCharTree(retioTree,nchar=10)

#same as number of nodes (minus root)
perfectParsCharTree(retioTree,nchar=12)

#more characters than the number of nodes
perfectParsCharTree(retioTree,nchar=20)
```

plotOccData — *Plotting Occurrence Data Across Taxa*

Description

plotOccData takes occurrence data which has been sorted into a by-taxon list, such as that output by taxonSortPBDBocc or may be output by simulations using sampleRanges and produces a plot showing the age uncertainty associated with individual occurrences, with occurrences of the same taxon grouped by color.

Usage

```
plotOccData(occList, groupLabel = NULL, occColors = NULL,
  lineWidth = NULL, xlims = NULL)
```

Arguments

occList	A list where every element is a table of occurrence data for a different taxon, such as that returned by taxonSortPBDBocc. The occurrence data can be either a two-column matrix composed of the lower and upper age bounds on each taxon occurrence, or has two named variables which match any of the field names given by the PBDB API under either the 'pbdb' vocab or 'com' (compact) vocab for early and late age bounds.
groupLabel	A character vector with a single string giving the name for the occurrence dataset used, such as the taxonomic name of the group examined. If not given (the default) a generic plot title is appended.
occColors	A vector of numbers or characters indicating colors on a color palette for use with basic plot. Must be the same length as occList. If empty, the default, the colors used are sampled randomly from the rainbow() function.
lineWidth	A numeric value giving the length to be used for the width of lines plotted in plotOccData. If not given (the default), this is calculated using an algorithm that selects an optimal line width for plotting.
xlims	A two element vector controlling the width of the horizontal time-scale the occurrence bars are plotted against. By default, this is not given and calculated internally.

Details

This function was originally conceived of in the following blog post: Link

Value

This function will invisibly return a list, with each per-taxon element containing the two-column matrix of age bounds for occurrences.

Author(s)

David W. Bapst

See Also

`taxonSortPBDBocc`, `occData2timeList` and the example graptolite dataset at `graptPBDB`

Examples

```
#load example graptolite PBDB occ dataset
data(graptPBDB)

#get formal genera
occSpecies<-taxonSortPBDBocc(graptOccPBDB, rank="species")

#plot it!
plotOccData(occSpecies)

#this isn't too many occurrences, because there are so few
    #formal grapt species in the PBDB

#genera is messier...

#get formal genera
occGenus<-taxonSortPBDBocc(graptOccPBDB, rank="genus")

#plot it!
plotOccData(occGenus)

#some of those genera have occurrences with very large
   #age uncertainties on them!
```

`plotTraitgram` *Plot a Traitgram for Continuous Traits*

Description

`plotTraitgram` plots a traitgram showing the evolution of a continuous trait. If node values are not given (i.e. the data is empirical data collected from tips, rather than simulated data), maximum-likelihood ancestral trait estimation is used to calculate node values. (Ackerly, 2009) given a tree and a set of continuous trait values.

Usage

```
plotTraitgram(trait, tree, main = "", conf.int = TRUE, lwd = 1.5)
```

Arguments

`trait`	A vector of continuous trait values. If the length of `trait` is equal to the number of tips and number of nodes, than it is presumed internal node values are given. If the length of `trait` is equal to the number of tips of `tree` then these will be treated as tip values and ancestral trait reconstruction will be used to reconstruct the missing ancestral values. If `trait` is not named, or if internal node values are given, then values will be presumed to be in the order of tip/node numbering in `tree$edge`, which for tips is the same as the ordering in `tree$tip.label`.
`tree`	A `phylo` object.
`main`	Main title of traitgram plot.
`conf.int`	If `TRUE` (the default), confidence intervals are plotted.
`lwd`	The line width used for branches in the figure.

Details

By default, this function will use `ace` from the library `ape` to reconstruct ancestral traits and confidence intervals using the PIC method, if internal node values (i.e. ancestral node values) are not given.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

Return no value, just plot the traitgram.

Note

One should probably never do ancestral trait estimation without looking at the confidence intervals, as these reconstructed estimates tend to be very uncertain.

Author(s)

David W. Bapst

References

Ackerly, D. 2009 Conservatism and diversification of plant functional traits: Evolutionary rates versus phylogenetic signal. *Proceedings of the National Academy of Sciences* **106**(Supplement 2):19699–19706.

See Also

`ace`

Also see the functions `traitgram` in the library picante and `phenogram` in the library phytools.

Examples

```
set.seed(444)
tree <- rtree(10)
trait <- rTraitCont(tree)

#first, traitgram without conf intervals
plotTraitgram(trait,tree,conf.int=FALSE)

#now, with
plotTraitgram(trait,tree)
#not much confidence, eh?

# plotting simulated data
    # with values for ancestral nodes as input
trait <- rTraitCont(tree, ancestor=TRUE)
plotTraitgram(tree=tree,trait=trait)
```

---

pqr2Ps *Joint Probability of A Clade Surviving Infinitely or Being Sampled Once*

Description

Given the rates of branching, extinction and sampling, calculates the joint probability of a random clade (of unknown size, from 1 to infinite) either (a) never going extinct on an infinite time-scale or (b) being sampled at least once, if it does ever go extinct. As we often assume perfect or close to perfect sampling at the modern (and thus we can blanket assume that living groups are sampled), we refer to this value as the Probability of Being Sampled, or simply P(s). This quantity is useful for calculating the probability distributions of waiting times that depend on a clade being sampled (or not).

Usage

```
pqr2Ps(p, q, r, useExact = TRUE)
```

Arguments

`p`	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.
`q`	Instantaneous rate of extinction (mu)
`r`	Instantaneous rate of sampling
`useExact`	If TRUE, an exact solution developed by Emily King is used; if FALSE, an iterative, inexact solution is used, which is somewhat slower (in addition to being inexact...).

Details

Note that the use of the word 'clade' here can mean a monophyletic group of any size, including a single 'species' (i.e. a single phylogenetic branch) that goes extinct before producing any descendants. Many scientists I have met reserve the word 'clade' for only groups that contain at least one branching event, and thus contain two 'species'. I personally prefer to use the generic term 'lineage' to refer to monophyletic groups of one to infinity members, but others reserve this term for a set of morphospecies that reflect an unbroken anagenetic chain.

Obviously the equation used makes assumptions about prior knowledge of the time-scales associated with clades being extant or not: if we're talking about clades that originated a short time before the recent, the clades that will go extinct on an infinite time-scale probably haven't had enough time to actually go extinct. On reasonably long time-scales, however, this infinite assumption should be reasonable approximation, as clades that survive 'forever' in a homogenous birth-death scenario are those that get very large immediately (similarly, most clades that go extinct also go extinct very shortly after originating... yes, life is tough).

Both an exact and inexact (iterative) solution is offered; the exact solution was derived in an entirely different fashion but seems to faithfully reproduce the results of the inexact solution and is much faster. Thus, the exact solution is the default. As it would be very simple for any user to look this up in the code anyway, here's the unpublished equation for the exact solution:

$$Ps = 1 - (((p + q + r) - (sqrt(((p + q + r)^2) - (4 * p * q))))/(2 * p))$$

Value

Returns a single numerical value, representing the joint probability of a clade generated under these rates either never going extinct or being sampled before it goes extinct.

Author(s)

This function is entirely the product of a joint effort between the package author (David W. Bapst), Emily A. King and Matthew W. Pennell. In particular, Emily King solved a nasty bit of calculus to get the inexact solution and later re-derived the function with a quadratic methodology to get the exact solution. Some elements of the underlying random walk model were provided by S. Nalayanan (a user on the website stackexchange.com) who assisted with a handy bit of math involving Catalan numbers.

References

Bapst, D. W., E. A. King and M. W. Pennell. In prep. Probability models for branch lengths of paleontological phylogenies.

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

See Also

`SamplingConv`

Examples

```
#with exact solution
pqr2Ps(p=0.1,q=0.1,r=0.1,useExact=TRUE)
#with inexact solution
pqr2Ps(p=0.1,q=0.1,r=0.1,useExact=TRUE)
```

---

probAnc *Probability of being a sampled ancestor of another sampled taxon*

Description

Uses models from Foote (1996) to calculate the probability

Usage

```
probAnc(p, q, R, mode = "budding", analysis = "directDesc", Mmax = 85,
  nrep = 10000)
```

Arguments

`p`	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.
`q`	Instantaneous rate of extinction (mu)
`R`	Per-interval probability of sampling a taxon at least once
`mode`	Mode of morphotaxon differentiation, based on definitions in Foote, 1996. Can be pure cladogenetic budding ("budding"), pure cladogenetic bifurcating ("bifurcating") or pure anagenetic within-lineage change ("anagenesis"; i.e. Foote's 'phyletic change'). Default mode is "budding".
`analysis`	The type of analysis to be performed, either the probability of sampling direct descendants (`"directDesc"`) or of sampling indirect descendants (`"indirectDesc"`).
`Mmax`	The maximum number of direct descendants (M) to sum over in the function, which is ideally meant to be a sum from zero to infinity, like nrep. Unfortunately, `(2*M)` is used in a factorial, which means we are limited to a relatively small upper bound on M.
`nrep`	Number of repetitions to run in functions which are meant to sum over infinity. Default is arbitrarily high.

Details

probAnc obtains the probability of sampling a descendant of a morphotaxon in the fossil record, given the sampling probability and estimates of origination and extinction rates. These values are always calculated assuming infinite time for the potential ancestor to produce daughter taxa (assuming it lives that long) and under homogenous birth, death and sampling rates/probabilities, which is a situation that may be overly ideal relative to many real fossil records.

This can be calculated for either direct descendants, i.e. the probability of sampling any morphotaxa that arise immediately from the particular morphotaxon that could be an ancestor, or indirect descendants, i.e. the probability for any morphotaxon that has the morphotaxon of question as an ancestor, no matter how distant. See the argument `analysis` for details. Mode of differentiation can also be varied for three different models, see the argument `mode`.

This probability is calculated including the probability that extinction might occur before any descendants are produced. Thus, if `p = q`, the probability of a taxon going extinct before it produces any descendants will be 0.5, which means that even when sampling is perfect (`R = 1`, meaning completeness of 100 can be no higher than 0.5. See Foote (1996) for a graphic depiction of this non-intuitive ceiling. For reasons (probably?) having to do with finite approximations of infinite summations, values close to perfect sampling may have values slightly higher than this ceiling, which is also apparent visually in the figures in Foote (1996). Thus, values higher than 0.5 when p=q should be discounted, and in general when sampling rate is high, results should be treated cautiously as overestimates.

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

See Also

`SamplingConv`

Examples

```
#examples, run at very low nrep for sake of speed (examples need to be fast)

#default: probability of sampling a direct descendant
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "budding", analysis="directDesc",nrep=100)

#other modes
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "bifurcating", analysis="directDesc",nrep=100)
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "anagenesis", analysis="directDesc",nrep=100)

#probability of having sampled indirect descendants of a taxon
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "budding", analysis="indirectDesc",nrep=100)
#default
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "bifurcating", analysis="indirectDesc",nrep=100)
probAnc(p = 0.1, q = 0.1, R = 0.5, mode = "anagenesis", analysis="indirectDesc",nrep=100)
```

`resolveTreeChar` *Resolve Polytomies Using Parsimony-Based Reconstruction of a Discrete Character*

Description

This function resolves a set of given topology with less than fully-binary phylogenetic resolution so that lineages are shifted and internal nodes added that minimize the number of independent character transitions needed to explain an observed distribution of discrete character states for the taxa on such a tree, under various maximum-parsimony algorithms of ancestral character reconstruction, powered ultimately by function `ancestral.pars` in library `phangorn`. This function is mainly designed for use with poorly resolved trees which are being assessed with the function `minCharChange`.

Usage

```
resolveTreeChar(tree, trait, orderedChar = FALSE, stateBias = NULL,
  iterative = TRUE, type = "MPR", cost = NULL, ambiguity = c(NA, "?"),
  dropAmbiguity = FALSE, polySymbol = "&", contrast = NULL)
```

Arguments

`tree`	A cladogram of type 'phylo'. Any branch lengths are ignored.
`trait`	A vector of trait values for a discrete character, preferably named with taxon names identical to the tip labels on the input tree.
`orderedChar`	Is the character of interest given for `trait` ordered or not? If FALSE (the default), then for each polytomy, all child nodes that appear to have the same state as the ancestor node will remain in the polytomy, and any additional states held by child nodes will each be grouped into their own unique polytomy that forms from a descendant node of the original polytomy. If TRUE, then the character will be reconstructed with a cost (step) matrix of a linear, ordered character, and polytomies will be resolved so that lineages with different states will be placed into a nested ladder that reflects the ordered character. As with the unordered option, child nodes with a state equivalent to the ancestral node will remain in the polytomy, while more primitive or more derived states will be sorted into their own separate ladders composed of paraphyletic groups, ordered so to move 'away' state-by-state from the ancestral node's inferred character state. This option is not applicable if `type = "ACCTRAN"`, as cost matrices cannot be used with ACCTRAN in `ancestral.pars`, and an error will be returned if `orderedChar=TRUE` but a cost matrix is given manually.
`stateBias`	This argument controls how `resolveTreeChar` handles ancestral node reconstructions that have multiple states competing for the maximum weight of any state (i.e. if states 0 and 1 both have 0.4 of the weight). The default, where `stateBias = NULL` causes uncertainty at nodes among states to be treated as a single 'group' identical to any states within it. Essentially, this means that for the example polytomy where the ancestor hax maximum weight for both 0 and 1, any child nodes with 0, 1 or both of these states will be considered to have an identical state for the purpose of grouping nodes for the purpose of further resolving polytomies. If and only if `orderedChar = TRUE`, then additional options of `stateBias = 'primitive'` and `stateBias = 'derived'` become available, which instead force uncertain node assignments to either be the most primitive (i.e. the minimum) or the most derived (i.e. the maximum) among the

	maximum-weight states. In particular, `stateBias = 'primitive'` should favor gains and bias any analysis of character transitions against finding reversals.
`iterative`	A logical argument which, if TRUE (the default), causes the function to repeat the polytomy-resolving functionality across the entire tree until the number of nodes stabilizes. If FALSE, polytomies are only passed a single time.
`type`	The parsimony algorithm applied by `ancestral.pars`, which can apply one of two: "MPR" (the default) is a relatively fast algorithm developed by Hamazawa et al. (1995) and Narushima and Hanazawa (1997), which relies on reconstructing the states at each internal node by re-rooting at that node. "ACCTRAN", the 'accelerated transitions' algorithm (Swofford and Maddison, 1987), favors character reversal over independent gains when there is ambiguity. The "ACCTRAN" option in ancestral.pars avoids repeated rerooting of the tree to search for a smaller set of maximum-parsimony solutions that satisfy the ACCTRAN algorithm, but does so by assigning edge weights. As of phangorn v1.99-12, both of these algorithms apply the Sankoff parsimony algorithm, which allows multifurcations (polytomies).
`cost`	A matrix of the cost (i.e. number of steps) necessary to change between states of the input character trait. If NULL (the default), the character is assumed to be unordered with equal cost to change from any state to another. Cost matrices only impact the "MPR" algorithm; if a cost matrix is given but `type = "ACCTRAN"`, an error is issued.
`ambiguity`	A vector of values which indicate ambiguous (i.e. missing or unknown) character state codings in supplied `trait` data. Taxa coded ambiguously as treated as being equally likely to be any state coding. By default, `NA` values and "?" symbols are treated as ambiguous character codings, in agreement with behavior of functions in packages `phangorn` and `Claddis`. This argument is designed to mirror an hidden argument with an identical name in function `phyDat` in package phangorn.
`dropAmbiguity`	A logical. If `TRUE` (which is not the default), all taxa with ambiguous codings as defined by argument `ambiguity` will be dropped prior to ancestral nodes being inferred. This may result in too few taxa.
`polySymbol`	A single symbol which separates alternative states for polymorphic codings; the default symbol is `"&"`, following the output by `Claddis`'s `ReadMorphNexus` function, where polymorphic taxa are indicated by default with a string with state labels separated by an `"&"` symbol. For example, a taxon coded as polymorphic for states 1 or 2, would be indicated by the string "1&2". `polySymbol` is used to break up these strings and automatically construct a fitting `contrast` table for use with this data, including for ambiguous character state codings.
`contrast`	A matrix of type integer with cells of 0 and 1, where each row is labelled with a string value used for indicating character states in `trait`, and each column is labelled with the formal state label to be used for assign taxa to particular character states. A value of 1 indicates that the respective coding string for that row should be interpreted as reflecting the character state listed for that column. A coding could reflect multiple states (such as might occur when taxa are polymorphic for some morphological character), so the sums of rows and columns can sum to more than 1. If `contrast` is not `NULL` (the default), the arguments will nullify This argument is designed to mirror an hidden argument with an identical

name in function phyDat in package phangorn. This structure is based on the contrasts tables used for statistical evaluation of factors. See the phangorn vignette "Special features of phangorn" for more details on its implementation within phangorn including an example. See examples below for the construction of an example contrast matrix for character data with polymorphisms, coded as character data output by Claddis's ReadMorphNexus function, where polymorphic taxa are indicated with a string with state labels separated by an "&" symbol.

Details

As shown in the example code below, this function offers a wide variety of options for manipulating the maximum-parsimony algorithm used (i.e. MPR versus ACCTRAN), the ordering (or not) of character states, and potential biasing of uncertainty character state reconstructions (when ordered characters are assessed). This allows for a wide variety of possible resolutions for a given tree with polytomies and a discrete character. In general, the author expects that use of this function will be optimal when applied to ordered characters using one of the stateBias options, perhaps `stateBias = "primitive"` (based on theoretical expectations for slow evolving characters). However, anecdotal use of this function with various simulation datasets suggests that the results are quite variable, and so the best option needs to be assessed based on the prior assumptions regarding the data and the performance of the dataset with the various arguments of this function.

Value

Returns the resulting tree, which may be fully resolved, partly more resolved or not more resolved at all (i.e. have less polytomies) depending on what was possible, as constrained by ambiguities in character reconstructions. Applying multi2di is suggested as a post-step to obtain a fully-resolved cladogram, if one is desired.

Author(s)

David W. Bapst

References

Hanazawa, M., H. Narushima, and N. Minaka. 1995. Generating most parsimonious reconstructions on a tree: A generalization of the Farris-Swofford-Maddison method. Discrete Applied Mathematics 56(2-3):245-265.

Narushima, H., and M. Hanazawa. 1997. A more efficient algorithm for MPR problems in phylogeny. Discrete Applied Mathematics 80(2-3):231-238.

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

Swofford, D. L., and W. P. Maddison. 1987. Reconstructing ancestral character states under Wagner parsimony. Mathematical Biosciences 87(2):199-229.

See Also

ancPropStateMat which is used internally by this function. This function was intentionally designed for use with minCharChange.

Examples

```
# let's write a quick&dirty ancestral trait plotting function

 quickAncPlot<-function(tree,trait,cex,orderedChar=FALSE,type="MPR",cost=NULL){
ancData<-ancPropStateMat(tree=tree, trait=trait, orderedChar=orderedChar)
ancCol<-(1:ncol(ancData))+1
  plot(tree,show.tip.label=FALSE,no.margin=TRUE,direction="upwards")
  tiplabels(pch=16,pie=ancData[(1:Ntip(tree)),],cex=cex,piecol=ancCol,
col=0)
  nodelabels(pie=ancData[-(1:Ntip(tree)),],cex=cex,piecol=ancCol)
  }

##########

# examples with simulated data

set.seed(2)
tree<-rtree(50)
#simulate under a likelihood model
trait<-rTraitDisc(tree,k=3,rate=0.7)
tree<-degradeTree(tree,prop_collapse=0.6)
tree<-ladderize(tree,right=FALSE)

#a bunch of type=MPR (default) examples
treeUnord<-resolveTreeChar(tree,trait,orderedChar=FALSE)
treeOrd<-resolveTreeChar(tree,trait,orderedChar=TRUE,stateBias=NULL)
treeOrdPrim<-resolveTreeChar(tree,trait,orderedChar=TRUE,stateBias="primitive")
treeOrdDer<-resolveTreeChar(tree,trait,orderedChar=TRUE,stateBias="derived")
#and finally an unordered one with ACCTRAN
treeACCTRAN<-resolveTreeChar(tree,trait,orderedChar=FALSE,type="ACCTRAN")

#compare number of nodes
Nnode(tree) #original
Nnode(treeUnord) #unordered, biasStates=NULL, MPR
Nnode(treeOrd) #ordered, biasStates=NULL
Nnode(treeOrdPrim) #ordered, biasStates='primitive'
Nnode(treeOrdDer) #ordered, biasStates='derived'
Nnode(treeACCTRAN) #unordered, biasStates=NULL, ACCTRAN

#let's compare original tree with unordered-resolved tree
layout(1:2)
quickAncPlot(tree,trait,orderedChar=FALSE,cex=0.3)
text(x=43,y=10,"Original",cex=1.5)
quickAncPlot(treeUnord,trait,orderedChar=FALSE,cex=0.3)
text(x=43,y=10,"orderedChar=FALSE",cex=1.5)
#some resolution gained

#now let's compare the original and ordered, both biasStates=NULL
layout(1:2)
```

```
quickAncPlot(tree,trait,orderedChar=FALSE,cex=0.3)
text(x=43,y=10,"Original",cex=1.5)
quickAncPlot(treeOrd,trait,orderedChar=TRUE,cex=0.3)
text(x=43,y=10,"orderedChar=TRUE",cex=1.5)

#now let's compare the three ordered trees
layout(1:3)
quickAncPlot(treeOrd,trait,orderedChar=TRUE,cex=0.3)
text(x=41,y=8,"ordered, biasStates=NULL",cex=1.5)
quickAncPlot(treeOrdPrim,trait,orderedChar=TRUE,cex=0.3)
text(x=41.5,y=8,"ordered, biasStates='primitive'",cex=1.5)
quickAncPlot(treeOrdDer,trait,orderedChar=TRUE,cex=0.3)
text(x=42,y=8,"ordered, biasStates='derived'",cex=1.5)

#let's compare unordered with ordered, biasStates='primitive'
layout(1:2)
quickAncPlot(treeUnord,trait,orderedChar=FALSE,cex=0.3)
text(x=41,y=8,"orderedChar=FALSE",cex=1.5)
quickAncPlot(treeOrdPrim,trait,orderedChar=TRUE,cex=0.3)
text(x=40,y=11,"orderedChar=TRUE",cex=1.5)
text(x=40,y=4,"biasStates='primitive'",cex=1.5)

#let's compare unordered with MPR to unordered with ACCTRAN
layout(1:2)
quickAncPlot(treeUnord,trait,orderedChar=FALSE,type="MPR",cex=0.3)
text(x=41,y=8,"unordered: MPR",cex=1.5)
quickAncPlot(treeACCTRAN,trait,orderedChar=FALSE,type="ACCTRAN",cex=0.3)
text(x=41,y=8,"unordered: ACCTRAN",cex=1.5)

#these comparisons will differ greatly between datasets
# need to try them on your own

layout(1)
```

`retiolitinae` *Cladogram and Range Data for the Retiolitinae*

Description

The majority rule consensus cladogram for 22 genera from the Retiolitinae, a clade of Silurian retiolitids, along with discrete time interval data taken from the same publication (Bates et al., 2005). Additional character state data are included for three major, binary-state morphological traits.

Format

This dataset is composed of three objects:

`retioTree` An ape 'phylo' object containing the consensus cladogram.

`retioRanges` A list containing two matrices. The first matrix describes the first and last interval times for 20 Silurian graptolite zones and the second matrix describes when the various genera on the cladogram first and last appear in those graptolite zones. (In other words, `retioRanges` has the 'timeList' format called by some paleotree functions).

`retioChar` A matrix containing binary presence-absence character states for these 22 Retiolitinae genera for three characters which they vary in: the presence of extrathecal threads (note only one taxon lacks this character state), the presence of determinant growth and the secondary loss of a nema via resorbtion. Note these character do not vary within these genera.

Details

Interval dates were taken from Sadler et al. (2009). These zones were not a 1-1 match to those in Bates et al., so it took some merging and splitting by the package author, so buyer beware.

Character data are from an in prep manuscript containing character data for certain major morphological innovations of graptoloids, coded for a large number of genera based on an extensive survey of the published descriptions. The character data presented here is a small subset of the full dataset.

Source

Source for cladogram and zonal ranges for genera:

Bates, D. E. B., A. Kozlowska, and A. C. Lenz. 2005. Silurian retiolitid graptolites: Morphology and evolution. *Acta Palaeontologica Polonica* 50(4):705-720.

Source for interval dates for graptolite zones:

Sadler, P. M., R. A. Cooper, and M. Melchin. 2009. High-resolution, early Paleozoic (Ordovician-Silurian) time scales. *Geological Society of America Bulletin* 121(5-6):887-906.

Source for morphological character data:

Collected for Bapst and Mitchell, in prep.

See Also

For more example graptolite datasets, see `graptDisparity`

Examples

```
#load data
data(retiolitinae)

#Can plot discrete time interval diversity curve with retioRanges
taxicDivDisc(retioRanges)

#Can plot the unscaled cladogram
plot(retioTree)
#Can plot the determinant growth character on the cladogram
tiplabels(pch=16,col=(retioChar[,2]+1),adj=0.25)

#Use basic time-scaling (terminal branches only go to FADs)
```

```
ttree<-bin_timePaleoPhy(tree=retioTree,timeList=retioRanges,type="basic",
ntrees=1,plot=TRUE)

#Note that this function creates stochastic time-scaled trees...
#A sample of 1 is not representative!

#phylogenetic diversity curve
phyloDiv(ttree)
```

---

reverseList *Reverse List Structure*

Description

Takes a list and reverses the list structure, such that list composed of five elements with eight sub-elements is restructured to have eight elements with five sub-elements each, with the order of elements and sub-elements being retained despite their reversal in hierarchical position.

Usage

```
reverseList(list, simplify = FALSE)
```

Arguments

`list`	A list composed of multiple elements, with each element a vector or list of equal length
`simplify`	Should the result be simplified, as the argument in sapply

Details

The function will fail and return an error if all sub-elements are not vectors or lists of equal length.

This function can be useful for instances when each element of a list is by-sample, composed of multiple, different tests on that sample, but where for further analysis/plotting, it would be beneficial to have a list where each element represented values from the same test performed across multiple samples (i.e. plotting a box-plot).

Value

Returns a list with a reversed structure relative to the input, see above.

Author(s)

David W. Bapst

Examples

```
list1<-list(list(1:3),list(1:3),list(1:3))
reverseList(list1,simplify=FALSE)
reverseList(list1,simplify=TRUE)
```

---

rootSplit *Split Tip Taxa by Root Divergence*

Description

Sorts terminal taxa into groups descended from each lineage splitting off of the root node.

Usage

```
rootSplit(tree)
```

Arguments

tree	A phylo object

Details

This function can be useful for studying the timing in the order of appearance of descended from different lineages descended from the first bifurcation.

Value

Returns a list with each element a character vector containing the names of terminal taxa descended from each lineage splitting off of the root node.

Author(s)

David W. Bapst

Examples

```
tree<-rtree(100)
rootSplit(tree)
```

`sampleRanges` *Sampling Taxon Ranges*

Description

A function for simulating the effect of incomplete sampling of the fossil record.

Usage

```
sampleRanges(taxad, r, alpha = 1, beta = 1, rTimeRatio = 1,
  modern.samp.prob = 1, min.taxa = 2, ranges.only = TRUE, minInt = 0.01,
  merge.cryptic = TRUE, randLiveHat = TRUE, alt.method = FALSE,
  plot = FALSE)
```

Arguments

`taxad`	A two-column matrix of per-taxon ranges. The five-column matrix output of `simFossilRecord`, post transformation with `fossilRecord2fossilTaxa` can also be supplied, which will be common in simulation usages.
`r`	Instantaneous average sampling rate per lineage time units; given as a vector of length one or length equal to the number of taxa
`alpha`	Alpha parameter of beta distribution; given as a vector of length one or length equal to the number of taxa
`beta`	Beta parameter of beta distribution; given as a vector of length one or length equal to the number of taxa
`rTimeRatio`	Ratio of most recent sampling rate over earliest sampling rate; given as a vector of length one or length equal to the number of taxa
`modern.samp.prob`	Probability of sampling living taxa at the present day (time=0), see below.
`min.taxa`	Minimum number of taxa sampled. The default is 2.
`ranges.only`	If TRUE, gives taxon first and last occurrences only. If FALSE, gives the time of all sampling events as a list.
`minInt`	Minimum interval size used for simulating complex models
`merge.cryptic`	If TRUE, sampling events for cryptic species will be merged into one taxon.
`randLiveHat`	If TRUE, taxa still alive at modern day have the end-point of their 'hat' chosen from a uniform distribution.
`alt.method`	If TRUE, use the alternative method of discretizing time even if a simple model of sampling is being simulated.
`plot`	If TRUE, plots the sampling models for each taxon against time.

Details

This function implements a range of sampling models in continuous time. Be default, sampling is simulated under the simplest model, where sampling occurs as a Poisson process under a instantaneous sampling rate (r) which is homogeneous through time and across lineages (Foote, 1997). Under this model, the waiting times to sampling events are exponentially distributed, with an average waiting time of 1/r. This useful property allows sampling to be rapidly simulated for many taxa under this simple model in sampleRanges, by repeatedly drawing waiting times between sampling events from an exponential distribution. This is the model that is run when alpha, beta and rTimeRatio are set to 1.

In addition to this simple model, sampleRanges also can consider a range of additional models, including the hatP and incP options of Liow et al. (2010). To describe the behavior of these models, users alter the default values for alpha, beta and rTimeRatio. These parameters, and r, can either be a single value which describes the behavior of the entire dataset or as a vector, of same length as the number of taxa, which describes the per-taxon value. When any rTimeRatio, alpha or beta value is not equal to one, then the sampling rate will vary across the duration of a taxon's temporal range. In general, setting alpha and beta equal to a value above 2 produce "hat" or bell-shaped curves where sampling rates peak at the midpoint of taxon ranges, while setting them unequal will produce asymmetric bell curves according to the beta function (Liow et al., 2010; Liow et al. set alpha=beta=4). rTimeRatio is the ratio of the sampling rate of the latest/most recent time divided by the earliest/oldest time.

The input r values will be interpreted differently based on whether one r value or per-taxon values were used. If one value was input, then it is assumed that r represent the grand mean r for the entire dataset for purposes of time-varying r, such that if rTimeRatio is not equal to 1, taxa near the end and start of the dataset will have very different per-taxon mean sampling rate. If per-taxon values of r were input, then each r is consider the per-taxon mean sampling rate. These will not be changed, but any within-lineage variation is distributed so that the mean is still the input per-taxon value. This also changes the interpretation of rTimeRatio, such that when a single r value and rTimeRatio is given, it is assumed the ratio describes the change in sampling rates from the start of the dataset to the end, while if multiple values are given for either r or rTimeRatio will instead see the value as describing the ratio at the first and last times of each taxon. For the pure hat model, this interpretation of 'r' as a grand mean sampling means that taxa will have a sampling rate of 2*r at the mid-peak of their range, which will have considerable implications for taxonomic incompleteness.

The particular distinctions about these parameter values are important: all models simulated in sampleRanges are structured to be effectively nested inside a most general model with parameters r, alpha, beta and rTimeRatio.

Note that the modeling of sampling in this function is independent and secondary of the actual simulation of the ranges, which are (generally) produced by the models of simFossilRecord with argument `r` (sampling rate) not set. Thus, 'hat-shaped range distributions' are only contained within single morphotaxa; they do not cross multiple morphotaxa in the case of anagenesis. Cryptic taxa each have their own hat and do not share a single hat; by default the ranges of cryptic taxa are merged to produce the range of a single observed morphotaxon.

'Hats' are constrained to start and end with a taxon's range, representing the rise and fall of taxa in terms of abundance and geographic range (Liow et al., 2010). However, for still-living taxa at the modern day, it is unknown how much longer they may be alive (for memoryless Poisson models, there is no age-dependent extinction). The treatment of these taxa with regards to their 'hat' (i. e. the beta distribution) is controlled by randLivehat: when ranLiveHat=FALSE, the beta distribution is fit so that the last appearance of still-alive taxa at the modern day is treated as a last appearance for calculating the hat. When TRUE, the default option, the still-alive taxa are considered to have gotten some distance between 0 and 1 through the beta distribution, as of the modern day. This point of progression is stochastically selected for each taxon by pulling a number from a uniform distribution, and used for calculating the hat.

Because sampling rate varies over morphotaxon ranges under any of these more complex models, sampling events cannot be quickly simulated as waiting times pulled from an exponential distribution. Instead, the taxon durations are discretized into a large number of small time intervals of length minInt (see above; minInt should be small enough that only one sampling event could feasibly happen per interval). The probability of an event occurring within each interval is calculated and used to stochastically simulate sampling events. For each interval, a number between 0 and 1 is randomly pulled from a uniform distribution and to the per-interval sampling probability to test if a sampling event occurred (if the random number is less than the probability, a sampling event is recorded). In general, this method is slower but otherwise comparable to the quicker waiting times method. See the examples below for a small test of this.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

If min.taxa is set to zero, the simulation may produce output in which no taxa were ever sampled.

If modern.samp.prob is set to 1.0 (the default), then living taxa will always be sampled at least at the present day (if there are any living taxa). If the probability is less than 1, they will be sampled with that probability at the modern day.

By default, this function will merge sampling events from morphologically cryptic taxa, listing them as occurrences for the earliest member of that group. To change this behavior, set merge.cryptic to FALSE.

Conditioning on sampling some minimum number of taxa may create strange simulation results for some analyses, such as simulation analyses of birth-death processes. Set min.taxa=0 to remove this conditioning.

Value

If ranges.only is TRUE, then the output is a two-column per-taxon matrix of first and last appearances in absolute time. NAs mean the taxon was never sampled in the simulation.

If ranges.only is FALSE (the default), the output is a list, where each element is a vector of sampling events the timing of sampling events, each corresponding to a different taxon in the input. Elements that are NA are unsampled taxa.

Author(s)

David W. Bapst

References

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Liow, L. H., T. B. Quental, and C. R. Marshall. 2010 When Can Decreasing Diversification Rates Be Detected with Molecular Phylogenies and the Fossil Record? *Systematic Biology* **59**(6):646–659.

See Also

simFossilRecord, binTimeData

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
layout(1:2)
#let's see what the 'true' diversity curve looks like in this case
taxicDivCont(taxa)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.5)
#plot the diversity curve based on the sampled ranges
taxicDivCont(rangesCont)
#compare the true history to what we might observe!

#let's try more complicated models!

#a pull-to-the-recent model with x5 increase over time similar to Liow et al.'s  incP
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r=0.5,rTimeRatio=5,plot=TRUE)
taxicDivCont(rangesCont1)

#a hat-shaped model
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r=0.5,alpha=4,beta=4,plot=TRUE)
taxicDivCont(rangesCont1)

#a combination of these
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r=0.5,alpha=4,beta=4,rTimeRatio=5,plot=TRUE)
taxicDivCont(rangesCont1)

#testing with cryptic speciation
layout(1)
recordCrypt<-simFossilRecord(p=0.1, q=0.1, prop.cryptic=0.5, nruns=1,
nTotalTaxa=c(20,30), nExtant=0)
taxaCrypt<-fossilRecord2fossilTaxa(recordCrypt)
rangesCrypt <- sampleRanges(taxaCrypt,r=0.5)
taxicDivCont(rangesCrypt)

#an example of hat-shaped models (beta distributions) when there are live taxa
set.seed(444)
recordLive<-simFossilRecord(p=0.1, q=0.05, nruns=1,
nTotalTaxa=c(5,100),nExtant=c(10,100))
taxaLive<-fossilRecord2fossilTaxa(recordLive)
#with end-points of live taxa at random points in the hat
rangesLive<-sampleRanges(taxaLive,r=0.1,alpha=4,beta=4,randLiveHat=TRUE,plot=TRUE)
#with all taxa end-points at end-point of hat
```

```
rangesLive<-sampleRanges(taxaLive,r=0.1,alpha=4,beta=4,randLiveHat=FALSE,plot=TRUE)



#simulate a model where sampling rate evolves under brownian motion
tree<-taxa2phylo(taxa,obs=taxa[,3])
sampRateBM <- rTraitCont(tree)
sampRateBM <- sampRateBM-min(sampRateBM)
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r=sampRateBM,plot=TRUE)
taxicDivCont(rangesCont1)

#evolving sampling rate, hat model and pull of the recent
layout(1:2)
rangesCont1 <- sampleRanges(taxa,r=sampRateBM,alpha=4,beta=4,rTimeRatio=5,plot=TRUE)
taxicDivCont(rangesCont1)
layout(1)

#the simpler model is simulated by pulling waiting times from an exponential
#more complicated models are simulated by discretizing time into small intervals
#are these two methods comparable?
#let's look at the number of taxa sampled under both methods
summary(replicate(100,sum(!is.na(sampleRanges(taxa,r=0.5,alt.method=FALSE)[,1]))))
summary(replicate(100,sum(!is.na(sampleRanges(taxa,r=0.5,alt.method=TRUE)[,1]))))
#they look pretty similar!
```

SamplingConv *Converting Sampling Estimates*

Description

Various functions for converting between estimates of sampling in the fossil record.

Usage

```
sProb2sRate(R, int.length = 1)

sRate2sProb(r, int.length = 1)

pqsRate2sProb(r, p, q, int.length = 1)

qsProb2Comp(R, q, p = NULL, mode = "budding", nrep = 10000)

qsRate2Comp(r, q)
```

Arguments

`R`	Per-interval probability of sampling a taxon at least once
`int.length`	Length of Time Intervals
`r`	Instantaneous rate of sampling
`p`	Instantaneous rate of speciation (lambda). If the underlying model assumed is anagenetic (e.g. taxonomic change within a single lineage, 'phyletic evolution') with no branching of lineages, then p will be used as the rate of anagenetic differentiation.
`q`	Instantaneous rate of extinction (mu)
`mode`	Mode of morphotaxon differentiation, based on definitions in Foote, 1996. Can be pure cladogenetic budding ("budding"), pure cladogenetic bifurcating ("bifurcating") or pure anagenetic within-lineage change ("anagenesis"; i.e. Foote's 'phyletic change'). Default mode is "budding".
`nrep`	Number of repetitions to run in functions which are meant to sum over infinity. Default is arbitrarily high.

Details

This is a family of functions which all convert from some estimate of sampling to another estimate of sampling. Some of these also require estimates of an rate associated with taxonomic diversification, such as the speciation/origination rate or extinction rate. Diversification rates used in these functions should always be the instantaneous rates, often called the per-capita rates by paleontologists (Foote, 2000).

As with many models used in the paleotree library, it is generally assumed that the fossil record of interest is composed of discrete relatively-static taxonomic units which diversify mainly by budding cladogenesis, and that sampling events are rare and approximated by a Poisson model of exponentially-distributed waiting times between sampling events. The veracity of those assumptions is difficult to test and the sensitivity of these analyses to relaxing those assumptions probably varies.

sProb2sRate and sRate2sProb give rough conversions for the probability of sampling once per time interval (R or "sProb" in this package as used in the references below) and the instantaneous rate of sampling per lineage/time unit ("sRate" or r). If you have estimates of the speciation and extinction rate, use pqsRate2sProb instead for a more accurate estimate of R.

qsProb2Comp and qsRate2Comp are different calculations for "Pp" or the probability/proportion of taxa sampled in a clade. Theoretically, one could use it to extrapolate out the 'true' diversity, assuming the sampling rate model was correct. (See Foote and Raup, 1996.)

See the references below for a more detailed explanation of the methods and formulae used. The relevant equations are generally found in the appendices of those papers.

Value

The converted sampling estimate, depending on the function used. See details above.

Author(s)

David W. Bapst, with advice from Michael Foote.

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

Foote, M. 1997 Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* **23**(3):278–300.

Foote, M. 2000 Origination and extinction components of taxonomic diversity: general problems. Pp. 74–102. In D. H. Erwin, and S. L. Wing, eds. Deep Time: Paleobiology's Perspective. The Paleontological Society, Lawrence, Kansas.

Foote, M., and D. M. Raup. 1996 Fossil preservation and the stratigraphic ranges of taxa. *Paleobiology* **22**(2):121–140.

Solow, A. R., and W. Smith. 1997 On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* **23**(3):271–277.

See Also

`sampleRanges`, `make_durationFreqDisc`, `make_durationFreqCont`, `probAnc`, `pqr2Ps`.

Examples

```
sRate2sProb(r=0.5)
sProb2sRate(R=0.1)
pqsRate2sProb(r=0.5,p=0.1,q=0.1)

# different modes can be tried
qsProb2Comp(R=0.1,q=0.1,mode="budding")
qsProb2Comp(R=0.1,q=0.1,mode="bifurcating")

qsRate2Comp(r=0.1,q=0.1)
```

`seqTimeList` *Construct a Stochastic Sequenced Time-List from an Unsequenced Time-List*

Description

This function randomly samples from a timeList object (i.e. a list composed of a matrix of interval start and end dates and a matrix of taxon first and last intervals), to find a set of taxa and intervals that do not overlap, output as a new timeList object.

Usage

```
seqTimeList(timeList, nruns = 100, weightSampling = FALSE)
```

Arguments

`timeList`	A list composed of two matrices, giving interval start and end dates and taxon first and last occurrences within those intervals. Some intervals are expected to overlap (thus necessitating the use of this function), and datasets lacking overlapping intervals will return an error message.
`nruns`	Number of new timeList composed of non-overlapping intervals produced.
`weightSampling`	If TRUE, weight sampling of new intervals toward smaller intervals. FALSE by default.

Details

Many analyses of diversification and sampling in the fossil record require a dataset composed of sequential non-overlappling intervals, but the nature of the geologic record often makes this difficult, with taxa from different regions, environments and sedimentary basins having first and last appearances placed in entirely in-congruent systems of chronostratigraphic intervals. While one option is to convert such occurrences to a single, global stratigraphic system, this may still result in overlapping intervals when fossil collections are poorly constrained stratigraphically. (For example, this may often be the case in global datasets.) This function offers an approach to avoid this issue in large datasets by randomly subsampling the available taxa and intervals to produce stochastic sets of ranges composed of data drawn from non-overlapping intervals.

This function is stochastic and thus should be set for many runs to produce many such solutions. Additionally, all solutions found are returned, and users may wish to sort amongst these to maximize the number of intervals and number of taxa returned. A single solution which maximizes returned taxa and intervals may not be a precise enough approach to estimating sampling rates, however, given the uncertainty in data. Thus, many runs should always be considered.

By default, solutions are searched for without consideration to the length of intervals used (i.e. the selection of intervals is 'unweighted'). Alternatively, we can 'weight' selection toward the smallest intervals in the set, using the argument `weightSampling`. Smaller intervals presumably overlap less and thus should retain more taxa and intervals of more equal length. However, in practise with empirical datasets, the package author finds these approaches do not seem to produce very different estimates.

For some datasets, many solutions found using seqTimeList may return infinite sampling values. This is often due to saving too many taxa found in single intervals to the exclusion of longer-ranging taxa (see the example). This excess of single interval taxa is a clear artifact of the randomized seqTimeList procedure and such solutions should probably be ignored.

Value

A list, composed of three elements: `nIntervals` which is a vector of the number of intervals in each solution, `nTaxa` which is a vector of the number of taxa in each solution and `timeLists` which is a list composed of each new timeList object as an element.

Author(s)

David W. Bapst

See Also

Resulting time-lists can be analyzed with `freqRat`, `durationFreq`, etc.

Additionally, `binTimeData` can be useful for simulating interval data.

Examples

```
#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(60,80), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges()
rangesCont <- sampleRanges(taxa,r=0.1)

#Now let's use binTimeData to get ranges in discrete overlapping intervals
    #via pre-set intervals input
presetIntervals <- cbind(c(1000,995,990,980,970,975,960,950,940,930,900,890,888,879,875),
  c(995,989,960,975,960,950,930,930,930,900,895,888,880,875,870))
rangesDisc1 <- binTimeData(rangesCont,int.times=presetIntervals)

seqLists<-seqTimeList(rangesDisc1,nruns=10)
seqLists$nTaxa
seqLists$nIntervals

#apply freqRat as an example analysis
sapply(seqLists$timeLists,freqRat)

#notice the zero and infinite freqRat estimates? What's going on?

freqRat(seqLists$timeLists[[4]],plot=TRUE)

#too few taxa of two or three interval durations for the ratio to work properly
    #perhaps ignore these estimates

#with weighted selection of intervals
seqLists<-seqTimeList(rangesDisc1,nruns=10,weightSampling=TRUE)

seqLists$nTaxa
seqLists$nIntervals
sapply(seqLists$timeLists,freqRat)

#didn't have much effect in this simulated example
```

---

simFossilRecord *Full-Scale Simulations of the Fossil Record with Birth, Death and Sampling of Morphotaxa*

Description

A complete birth-death-sampling branching simulator that captures morphological-taxon identity of lineages, as is typically discussed in models of paleontological data. This function allows for the use of precise point constraints to condition simulation run acceptance and can interpret complex character strings given as rate values for use in modeling complex processes of diversification and sampling.

Usage

```
simFossilRecord(p, q, r = 0, anag.rate = 0, prop.bifurc = 0,
  prop.cryptic = 0, modern.samp.prob = 1, startTaxa = 1, nruns = 1,
  maxAttempts = Inf, totalTime = c(0, 1000), nTotalTaxa = c(1, 1000),
  nExtant = c(0, 1000), nSamp = c(0, 1000), tolerance = 10^-4,
  maxStepTime = 0.01, shiftRoot4TimeSlice = "withExtantOnly",
  count.cryptic = FALSE, negRatesAsZero = TRUE, print.runs = FALSE,
  sortNames = FALSE, plot = FALSE)
```

Arguments

`p, q, r, anag.rate`
: These parameters control the instantaneous ('per-capita') rates of branching, extinction, sampling and anagenesis, respectively. These can be given as a number equal to or greater than zero, or as a character string which will be interpreted as an algebraic equation. These equations can make use of three quantities which will/may change throughout the simulation: the standing richness is `N`, the current time passed since the start of the simulation is `T`, the present duration of a given still-living lineage since its origination time is codeD, and the current branching rate is `P` (corresponding to the argument name `p`). Note that `P` cannot be used in equations for the branching rate itself; it is for making other rates relative to the branching rate. By default, the rates `r` and `anag.rate` are set to zero, so that the default simulator is a birth-death simulator. Rates set to `= Inf` are treated as if 0. When a rate is set to 0, this event type will not occur in the simulation. Setting certain processes to zero, like sampling, may increase simulation efficiency, if the goal is a birth-death or pure-birth model. See documentation for argument `negRatesAsZero` about the treatment of rates that decrease below zero. Notation of branching, extinction and sampling rates as `p, q, r` follows what is typical for the paleobiology literature (e.g. Foote, 1997), not the Greek letters `lambda, mu, phi` found more typically in the biological literature (e.g. Stadler, 2009; Heath et al., 2014; Gavryushkina et al., 2014).

`prop.cryptic, prop.bifurc`
: These parameters control (respectively) the proportion of branching events that have morphological differentiation, versus those that are cryptic (`prop.cryptic`) and the proportion of morphological branching events that are bifurcating, as opposed to budding. Both of these proportions must be a number between 0 and 1. By default, both are set to zero, meaning all branching events are events of budding cladogenesis. See description of the available models of morphological differentiation in the *Description* section.

`modern.samp.prob`
: The probability that a taxon is sampled at the modern time (or, for `timeSliceFossilRecord`, the time at which the simulation data is slice). Must be a number between 0 and 1. If 1, all taxa that survive to the modern day (to the `sliceTime`) are sampled, if 0, none are.

`startTaxa`
: Number of initital taxa to begin a simulation with. All will have the simulation start date listed as their time of origination.

`nruns`
: Number of simulation datasets to accept, save and output. If `nruns = 1`, output will be a single object of class `fossilRecordSimulation`, and if `nruns` is greater than 1, a list will be output composed of `nruns` objects of class `fossilRecordSimulation`.

`maxAttempts`
: Number of simulation attempts allowed before the simulation process is halted with an error message. Default is `Inf`.

`totalTime, nTotalTaxa, nExtant, nSamp`
: These arguments represent stopping and acceptance conditions for simulation runs. They are respectively `totalTime`, the total length of the simulation in time-units, `nTotalTaxa`, the total number of taxa over the past evolutionary history of the clade, `nExtant`, the total number of extant taxa at the end of the simulation and `nSamp` the total number of sampled taxa (not counting extant taxa sampled at the modern day). These are used to determine when to end simulation runs, and whether to accept or reject them as output. They can be input as a vector of two numbers, representing minimum and maximum values of a range for accepted simulation runs (i.e. the simulation length can be between 0 and 1000 time-steps, by default), or as a single number, representing a point condition (i.e. if `nSamp = 100` then the only simulation runs with exactly 100 taxa sampled will be output). Note that it is easy to set combinations of parameters and run conditions that are impossible to produce satisfactory input under, in which case `simFossilRecord` would run in a nonstop loop. How cryptic taxa are counted for the sake of these conditions is controlled by argument `count.cryptic`.

`tolerance`
: A small number which defines a tiny interval for the sake of placing run-sampling dates before events and for use in determining whether a taxon is extant in simFossilRecordMethods.

`maxStepTime`
: When rates are time-dependent (i.e. when parameters 'D' or 'T' are used in equations input for one of the four rate arguments), then protocol used by `simFossilRecord` of drawing waiting times to the next event could produce a serious mismatch of resulting process to the defined model, because the possibility of new events is only considered at the end of these waiting times. Instead, any time a waiting time greater than `maxStepTime` is selected, then instead *no* event occurs and a time-step equal to `maxStepTime` occurs instead, thus effectively discretizing the progression of time in the simulations run by `simFossilRecord`. Decreasing this value will increase accuracy (as the time-scale is effectively more discretized) but increase computation time, as the computer will need to stop and check rates to see if an event happened more often. Users should toggle this value relative to the time-dependent rate equations they input, relative to the rate of change in rates expected in time-dependent rates.

`shiftRoot4TimeSlice`	Should the dating of events be shifted, so that the date given for `sliceTime` is now 0, or should the dates not be shifted, so that they remain on the same scale as the input? This argument accepts a logical TRUE or FALSE, but also accepts the string `"withExtantOnly"`, which will only 'shift' the time-scale if living taxa are present, as determined by having ranges that overlap within `tolerance` of `sliceTime`.
`count.cryptic`	If TRUE, cryptic taxa are counted as separate taxa for conditioning limits that count a number of taxon units, such as `nTotalTaxa`, `nExtant` and `nSamp`. If FALSE (the default), then each cryptic complex (i.e. each distinguishable morphotaxon) is treated as a single taxon. See examples.
`negRatesAsZero`	A logical. Should rates calculated as a negative number cause the simulation to fail with an error message (`= FALSE`) or should these be treated as zero (`"= TRUE"`, the default). This is equivalent to saying that the `rate.as.used = max(0, rate.as.given)`.
`print.runs`	If TRUE, prints the proportion of simulations accepted for output to the terminal.
`sortNames`	If TRUE, output taxonomic lists are sorted by the taxon names (thus sorting cryptic taxa together) rather than by taxon ID number (i.e. the order they were simulated in).
`plot`	If TRUE, plots the diversity curves of accepted simulations, including both the diversity curve of the true number of taxa and the diversity curve for the 'observed' (sampled) number of taxa.

Details

`simFossilRecord` simulates a birth-death-sampling branching process (ala Foote, 1997, 2000; Stadler, 2009) in which lineages of organisms may branch, go extinct or be sampled thought a continuous time-interval, with the occurrence of these events modeled as Poisson process controlled by some set of instantaneous rates. This model is ultimately based on the birth-death model (Kendall, 1948; Nee, 2006), which is widely implemented in many R packages. Unlike other such typical branching simulators, this function enmeshes the lineage units within explicit models of how lineages are morphologically differentiated (Bapst, 2013). This is key to allow comparison to datasets from the fossil record, as morphotaxa are the basic units of paleontological diversity estimates and phylogenetic analyses.

Models of Morphological Differentiation and Branching (Cladogenesis and Anagenesis)

These models of morphological differentiation do not involve the direct simulation of morphological traits. Instead, morphotaxon identity is used as a proxy of the distinctiveness of lineages on morphological grounds, as if there was some hypothetical paleontologist attempting to taxonomically sort collections of specimens of these simulated lineages. Two lineages are either identical, and thus share the same morphotaxon identity, or they are distinct, and thus have separate morphotaxon identities. Morphological differentiation is assumed to be an instantaneous process for the purposes of this model, such that no intermediate could be uncovered.

Specifically, `simFossilRecord` allows for three types of binary branching events (here grouped under the term 'cladogenesis': 'budding cladogenesis', 'bifurcating cladogenesis' and 'cryptic cladogenesis', as well as for a fourth event-type, 'anagenesis' (see Wagner and Erwin, 1995; Foote, 1996, and Bapst, 2013, for further details). Budding, bifurcation and cryptic cladogenetic events all share in common that a single geneological lineage splits into two descendant lineages, but differ in the morphological differentiation of these child lineages relative to their parent. Under budding cladogenesis, only one of the child lineages becomes morphologically distinguishable from the parent, and thus the ancestral morphotaxon persists through the branching event as the un-differtiated child lineage. Under bifurcating cladogenesis, both child lineages become immediately distinct from the ancestor, and thus two new morphotaxa appear while the ancestor terminates in an event known as 'pseudoextinction'. Crytic cladogenesis has no morphological differentiation: both child lineages are presumed to be indistinct from the ancestor and from each other, which means a hypothetical differentiation independent of any branching, such that a morphotaxon instanteously transitions to a new morphotaxon identity, resulting in the pseudoextinction of the ancestral morphotaxon and the immediate 'pseudospeciation' of the child morphotaxon. The two morphotaxa do not overlap in time at all, as modeled here (contra to the models described by Ezard et al., 2012). For ease of following these cryptic lineages, cryptic cladogenetic events are treated in terms of data structure similarly to budding cladogenetic events, with one child lineage treated as a persistance of the ancestral lineage, and the other as a new morphologically indistinguishable lineage. This model of cryptic cladogenesis is ultimately based on the hierarchical birth-death model used by many authors for modeling patterns across paraphyletic higher taxa and the lower taxon units within them (e.g. Patzkowsky, 1995; Foote, 2012).

The occurrence of the various models is controlled by multiple arguments of simFossilRecord. The overall instantaneous rate of branching (cladogenesis) is controlled by argument p, and the proportion of each type of cladogenesis controlled by arguments prop.bifurc and prop.cryptic. prop.cryptic controls the overall probability that any branching event will be cryptic versus involving any morphological differentiation (budding or bifurcating). If prop.cryptic = 1, all branching events will be cryptic cladogenesis, and if prop.cryptic = 0, all branching events will involve morphological differentiation and none will be cryptic. prop.bifurc controls how many branching events that involve morphological differentiation (i.e. the inverse of prop.cryptic) are bifurcating, as opposed to budding cladogenesis. If prop.bifurc = 1, all morphologically-differentiating branching events will be bifurcating cladogenesis, and if prop.bifurc = 0, all morphologically-differentiating branching events will be budding cladogenesis. Thus, for example, the probability of a given cladogenesis event being budding is given by:

```
Prob(budding cladogenesis at a branching event) = (1 - prop.cryptic) * (1 - prop.bifurc)
```

By default, prop.cryptic = 0 and prop.bifurc = 0, so all branching events by default will be instances of budding cladogenesis. Anagenesis is completely independent of these, controlled as its own Poisson process with an instantaneous rated defined by the argument anag.rate. By default, this rate is set to zero and thus there is no anagenetic events without user intervention.

Stopping Conditions and Acceptance Criteria for Simulations

How forward-time simulations are generated, halted and whether they are accepted or not for output is a critical component of simulation design. Most uses of simFossilRecord will involve iteratively generating and analyzing multiple simulation runs. Runs are only accepted for output if they meet the conditioning criteria defined in the arguments, either matching point constraints or falling within range constraints. However, this requires separating the processes of halting simulation runs and accepting a run for output, particularly to avoid bias related to statistical sampling issues.

Hartmann et al. (2011) recently discovered a potential statistical artifact when branching simulations are conditioned on some number of taxa. Previously within paleotree, this was accounted for in the deprecated function simFossilTaxa by a complex arrangement of minimum and maximum constraints, and an (incorrect) presumption that allowing simulations to continue for a short distance after constraints were reached. This strategy is not applied here. Instead, simFossilRecord applies the General Sampling Algorithm presented by Hartmann et al. (or at least, a close variant). A simulation continues until extinction or some maximum time-constraint is reached, evaluated for intervals that match the set run conditions (e.g. `nExtant`, `nTotalTime`) and, if some interval or set of intervals matches the run conditions, a date is randomly sampled from within this interval/intervals. The simulation is then cut at this date using the `timeSliceFossilRecord` function, and saved as an accepted run. The simulation data is otherwise discarded and then a new simulation initiated (thus, at most, only one simulated dataset is accepted from one simulation run).

Thus, accepted simulations runs should reflect unbiased samples of evolutionary histories that precisely match the input constraints, which can be very precise, unlike how stopping and acceptance conditions were handled in the previous (deprecated) `simFossilTaxa` function. Of course, selecting very precise constraints that are very unlikely or impossible given other model parameters may take considerable computation time to find acceptable simulation runs, or effectively never find any acceptable simulation runs.

On Time-Scale Used in Output

Dates given in the output are on an reversed absolute time-scale; i.e. time decreases going from the past to the future, as is typical in paleontological uses of time (as time before present) and as for most function in package `paleotree`. The endpoints of the time-scale are decided by details of the simulation and can be modified by several arguments. By default (with `shiftRoot4TimeSlice = "withExtantOnly"`), any simulation run that is accepted with extant taxa will have zero as the *end-time* (i.e. when those taxa are extant), as zero is the typical time assigned to the modern day in empirical studies. If a simulation ends with all taxa extinct, however, then instead the *start-time* of a run (i.e. when the run initiates with starting taxa) will be maximum value assigned to the conditioning argument `totalTime`. If `shiftRoot4TimeSlice = FALSE` then the *start-time* of the run will always be this maximum value for `totalTime`, and any extant taxa will stop at some time greater than zero.

Value

`simFossilRecord` returns either a single object of class `fossilRecordSimulation` or a list of multiple such objects, depending on whether `nruns` was 1 or more.

An object of class `fossilRecordSimulation` consists of a list object composed of multiple elements, each of which is data for 'one taxon'. Each data element for each taxon is itself a list, composed of two elements: the first describes vital information about the taxon unit, and the second describes the sampling times of each taxon.

The first element of the list (named `$taxa.data`) is a distinctive six-element vector composed of numbers (some are nominally integers, but not all, so all are stored as double-precision integers) with the following field names:

`taxon.id` The ID number of this particular taxon-unit.

`ancestor.id` The ID number of the ancestral taxon-unit. The initial taxa in a simulation will be listed with `NA` as their ancestor.

`orig.time` True time of origination for a taxon-unit in absolute time.

`ext.time` True time of extinction for a taxon-unit in absolute time. Extant taxa will be listed with an `ext.time` of the run-end time of the simulation run, which for simulations with extant taxa is 0 by default (but this may be modified using argument `shiftRoot4TimeSlice`).

`still.alive` Indicates whether a taxon-unit is 'still alive' or not: '1' indicates the taxon-unit is extant, '0' indicates the taxon-unit is extinct `looks.like` The ID number of the first morphotaxon in a dataset that 'looks like' this taxon-unit; i.e. belongs to the same multi-lineage cryptic complex. Taxa that are morphologically distinct from any previous lineage will have their `taxon.id` match their `looks.like`. Thus, this column is rather uninformative unless cryptic cladogenesis occurred in a simulation.

The second element for each taxon-unit is a vector of sampling times, creatively named `$sampling.times`, with each value representing a data in absolute time when that taxon was sampled in the simulated fossil record. If a taxon was never sampled, this vector is an empty numeric vector of `length = 0`.

As is typical for paleontological uses of absolute time, absolute time in these simulations is always decreasing toward the modern; i.e. an absolute date of 50 means a point in time which is 50 time-units before the present-day, if the present-day is zero (the default, but see argument `shiftRoot4TimeSlice`).

Each individual element of a `fossilRecordSimulation` list object is named, generally of the form "t1" and "t2", where the number is the `taxon.id`. Cryptic taxa are instead named in the form of "t1.2" and "t5.3", where the first number is the taxon which they are a cryptic descendant of (`looks.like`) and the second number, after the period, is the order of appearance of lineage units in that cryptic complex. For example, for "t5.3", the first number is the `taxon.id` and the second number communicates that this is the third lineage to appear in this cryptic complex.

Author(s)

David W. Bapst, inspired by code written by Peter Smits.

References

Bapst, D. W. 2013. When Can Clades Be Potentially Resolved with Morphology? *PLoS ONE* 8(4):e62312.

Ezard, T. H. G., P. N. Pearson, T. Aze, and A. Purvis. 2012. The meaning of birth and death (in macroevolutionary birth-death models). *Biology Letters* 8(1):139-142.

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141–151.

Foote, M. 1997. Estimating Taxonomic Durations and Preservation Probability. *Paleobiology* 23(3):278-300.

Foote, M. 2000. Origination and extinction components of taxonomic diversity: general problems. Pp. 74-102. In D. H. Erwin, and S. L. Wing, eds. *Deep Time: Paleobiology's Perspective.* The Paleontological Society, Lawrence, Kansas.

Foote, M. 2012. Evolutionary dynamics of taxonomic structure. *Biology Letters* 8(1):135-138.

Gavryushkina, A., D. Welch, T. Stadler, and A. J. Drummond. 2014. Bayesian Inference of Sampled Ancestor Trees for Epidemiology and Fossil Calibration. *PLoS Comput Biol.* 10(12):e1003919.

Hartmann, K., D. Wong, and T. Stadler. 2010 Sampling Trees from Evolutionary Models. *Systematic Biology* **59**(4):465–476.

Heath, T. A., J. P. Huelsenbeck, and T. Stadler. 2014. The fossilized birth-death process for coherent calibration of divergence-time estimates. *Proceedings of the National Academy of Sciences* 111(29):E2957-E2966.

Kendall, D. G. 1948 On the Generalized "Birth-and-Death" Process. *The Annals of Mathematical Statistics* **19**(1):1–15.

Nee, S. 2006 Birth-Death Models in Macroevolution. *Annual Review of Ecology, Evolution, and Systematics* **37**(1):1–17.

Patzkowsky, M. E. 1995. A Hierarchical Branching Model of Evolutionary Radiations. *Paleobiology* 21(4):440-460.

Solow, A. R., and W. Smith. 1997 On Fossil Preservation and the Stratigraphic Ranges of Taxa. *Paleobiology* **23**(3):271–277.

Stadler, T. 2009. On incomplete sampling under birth-death models and connections to the sampling-based coalescent. *Journal of Theoretical Biology* 261(1):58-66.

Wagner, P. J., and D. H. Erwin. 1995. Phylogenetic patterns as tests of speciation models. Pp. 87-122. In D. H. Erwin, and R. L. Anstey, eds. *New approaches to speciation in the fossil record.* Columbia University Press, New York.

See Also

`simFossilRecordMethods`

This function essentially replaces and adds to all functionality of the deprecated `paleotree` functions `simFossilTaxa`, `simFossilTaxaSRCond`, `simPaleoTrees`, as well as the combined used of `simFossilTaxa` and `sampleRanges` for most models of sampling.

Examples

```
set.seed(2)

# quick birth-death-sampling run with 1 run, 50 taxa

record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
nTotalTaxa=50, plot=TRUE)


# examining multiple runs of simulations

#example of repeated pure birth simulations over 50 time-units
records <- simFossilRecord(p=0.1, q=0, nruns=10,
totalTime=50, plot=TRUE)
#plot multiple diversity curves on a log scale
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE,plotLogRich=TRUE)
#histogram of total number of taxa
hist(sapply(records,nrow))

#example of repeated birth-death-sampling simulations over 50 time-units
records <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=10,
totalTime=50, plot=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE)

#like above, but conditioned instead on having 10 extant taxa
# between 1 and 100 time-units
set.seed(4)
```

```
records <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=10,
totalTime=c(1,300), nExtant=10, plot=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE)

################################################

# How probable were the runs I accepted?
# The effect of conditions

set.seed(1)

# Let's look at an example of a birth-death process
# with high extinction relative to branching
# use default run conditions (barely any conditioning)
# use print.runs to look at acceptance probability
records <- simFossilRecord(p=0.1, q=0.8, nruns=10,
print.runs=TRUE, plot=TRUE)
# 10 runs accepted from a total of 10 !

# now let's give much more stringent run conditions
# require 3 extant taxa at minimum, 5 taxa total minimum
records <- simFossilRecord(p=0.1, q=0.8, nruns=10,
nExtant=c(3,100), nTotalTaxa=c(5,100),
print.runs=TRUE, plot=TRUE)
# thousands of simulations to just obtail 10 accepable runs!
# most ended in extinction before minimums were hit

# beware analysis of simulated where acceptance conditions
# are too stringent: your data will be a 'special case'
# of the simulation parameters
# it will also take you a long time to generate reasonable
# numbers of replicates for whatever analysis you are doing

# TLDR: You should look at print.runs=TRUE

######################

# Using the rate equation-input for complex diversification models

# First up... Diversity Dependent Models!
# Let's try Diversity-Dependent Branching over 50 time-units

# first, let's write the rate equation
# We'll use the diversity dependent rate equation model
# from Ettienne et al. 2012 as an example here
# Under this equation, p=q at carrying capacity K
# Many others are possible!
# Note that we don't need to use max(0,rate) as negative rates
# are converted to zero by default, as controlled by
# the argument negRatesAsZero

# From Ettiene et al.
# lambda = lambda0 - (lambda0 - mu)*(n/K)
# lambda and mu are branching rate and extinction rate
# lambda and mu == p and q in paleotree (i.e. Foote convention)
# lambda0 is the branching rate at richness=0
# K is the carrying capacity
# n is the richness

# 'N' is the algebra symbol for standing taxonomic richness
# for simFossilRecord's simulation capabilities
# also branching rate cannot reference extinction rate
# we'll have to set lambda0, mu and K in the rate equation directly

lambda0 <- 0.3 # branching rate at 0 richness in Ltu
K <- 40 # carrying capacity
mu <- 0.1 # extinction rate will 0.1 Ltu (= 1/3 of lambda0)

# technically, mu here represents the lambda at richness=K
# i.e. lambdaK
# Ettienne et al. are just implicitly saying that the carrying capacity
# is the richness at which lambda==mu

# construct the equation programmatically using paste0
branchingRateEq<-paste0(lambda0,"-(",lambda0,"-",mu,")*(N/",K,")")
# and take a look at it...
branchingRateEq
# its a thing of beauty, folks

# now let's try it
records <- simFossilRecord(p=branchingRateEq, q=mu, nruns=3,
totalTime=100, plot=TRUE, print.runs=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE)
# those are some happy little diversity plateaus!


# now let's do diversity-dependent extinction

# let's slightly modify the model from Ettiene et al.
# mu = mu0 + (mu0 - muK)*(n/K)

mu0<-0.001 # mu at n=0
muK<-0.1 # mu at n=K (should be equal to lambda at K)
K<-40
lambda<-muK # equal to muK

# construct the equation programmatically using paste0
extRateEq<-paste0(mu0,"-(",mu0,"-",muK,")*(N/",K,")")
extRateEq

# now let's try it
records <- simFossilRecord(p=lambda, q=extRateEq, nruns=3,
totalTime=100, plot=TRUE, print.runs=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
```

```
multiDiv(records,plotMultCurves=TRUE)

# these plateaus looks a little more spiky
#( maybe there is more turnover at K? )
# also, it took a longer for the rapid rise to occur

# Now let's try an example with time-dependent origination
# and extinction constrained to equal origination

# Note! Use of time-dependent parameters "D" and "T" may
# result in slower than normal simulation run times
# as the time-scale has to be discretized; see
# info for argument maxTimeStep above

# First, let's define a time-dependent rate equation
# "T" is the symbol for time passed
timeEquation<-"0.4-(0.007*T)"

#in this equation, 0.4 is the rate at time=0
# and it will decrease by 0.007 with every time-unit
# at time=50, the final rate will be 0.05
# We can easily make it so extinction is always equal to branching rate
# "P" is the algebraic equivalent for "branching rate" in simFossilRecord

# now let's try it
records <- simFossilRecord(p=timeEquation, q="P", nruns=3,
totalTime=50, plot=TRUE, print.runs=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE)
# high variability that seems to then smooth out as turnover decreases

# And duration what about duration-dependent processes?
# let's do a duration-dep extinction equation:
durDepExt<-"0.01+(0.01*D)"

# okay, let's take it for a spin
records <- simFossilRecord(p=0.1, q=durDepExt, nruns=3,
totalTime=50, plot=TRUE, print.runs=TRUE)
records<-lapply(records,fossilRecord2fossilTaxa)
multiDiv(records,plotMultCurves=TRUE)
# creates runs full of short lived taxa

##########################################################

# Speciation Modes
# Some examples of varying the 'speciation modes' in simFossilRecord

# The default is pure budding cladogenesis
# anag.rate = prop.bifurc = prop.cryptic = 0
# let's just set those for the moment anyway
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0,
nruns=1, nTotalTaxa=c(20,30) ,nExtant=0, plot=TRUE)
```

```
#convert and plot phylogeny
# note this will not reflect the 'budding' pattern
# branching events will just appear like bifurcation
# its a typical convention for phylogeny plotting
converted<-fossilRecord2fossilTaxa(record)
tree<-taxa2phylo(converted,plot=TRUE)

#now, an example of pure bifurcation
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=1, prop.cryptic=0,
nruns=1, nTotalTaxa=c(20,30) ,nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)

# all the short branches are due to ancestors that terminate
# via pseudoextinction at bifurcation events

# an example with anagenesis = branching
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0.1, prop.bifurc=0, prop.cryptic=0,
nruns=1, nTotalTaxa=c(20,30) ,nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)
# lots of pseudoextinction

# an example with anagenesis, pure bifurcation
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0.1, prop.bifurc=1, prop.cryptic=0,
nruns=1, nTotalTaxa=c(20,30) ,nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)
# lots and lots of pseudoextinction

# an example with half cryptic speciation
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0.5,
nruns=1, nTotalTaxa=c(20,30), nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)

# notice that the tree has many more than the maximum of 30 tips:
# that's because the cryptic taxa are not counted as
# separate taxa by default, as controlled by count.cryptic

# an example with anagenesis, bifurcation, cryptic speciation
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0.1, prop.bifurc=0.5, prop.cryptic=0.5,
nruns=1, nTotalTaxa=c(20,30), nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)
# note in this case, 50% of branching is cryptic
# 25% is bifurcation, 25% is budding

# an example with anagenesis, pure cryptic speciation
# morphotaxon identity will thus be entirely indep of branching!
# I wonder if this is what is really going on, sometimes...
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0.1, prop.bifurc=0, prop.cryptic=1,
nruns=1, nTotalTaxa=c(20,30), nExtant=0)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record),plot=TRUE)

# merging cryptic taxa when all speciation is cryptic
set.seed(1)
record <- simFossilRecord(p=0.1,
q=0.1, r=0.1,
prop.crypt=1,
totalTime=50, plot=TRUE)
# there looks like there is only a single taxon, but...
length(record) #actual number of cryptic lineages

#############

# playing with count.cryptic with simulations of pure cryptic speciation

#can choose to condition on total morphologically-distinguishable taxa
    #or total taxa including cryptic taxa with count.cryptic=FALSE

# an example with pure cryptic speciation with count.cryptic=TRUE
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=1,
nruns=1, totalTime=50, nTotalTaxa=c(10,100), count.cryptic=TRUE)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record))
plot(tree);axisPhylo()
# notice how the tip labels indicate all are the same morphotaxon

# we'll replace the # of taxa constraints with a time constraint
# or else the count.cryptic=FALSE simulation will never end!

# an example with pure cryptic speciation with count.cryptic=FALSE
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=1,
nruns=1, totalTime=50, count.cryptic=FALSE)
tree<-taxa2phylo(fossilRecord2fossilTaxa(record))
plot(tree);axisPhylo()

#let's look at numbers of taxa returned when varying count.cryptic
# with prop.cryptic=0.5

#simple simulation going for 50 total taxa

#first, count.cryptic=FALSE (default)
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0.5,
nruns=1, nTotalTaxa=50, count.cryptic=FALSE)
taxa<-fossilRecord2fossilTaxa(record)
nrow(taxa)                  #number of lineages (inc. cryptic)
length(unique(taxa[,6]))            #number of morph-distinguishable taxa

# and count.cryptic=TRUE
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0.5,
nruns=1, nTotalTaxa=50, count.cryptic=TRUE)
taxa<-fossilRecord2fossilTaxa(record)
nrow(taxa)                  #number of lineages (inc. cryptic)
length(unique(taxa[,6]))            #number of morph-distinguishable taxa

# okay...
# now let's try with 50 extant taxa

#first, count.cryptic=FALSE (default)
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0.5,
nruns=1, nExtant=10, totalTime=c(1,100), count.cryptic=FALSE)
taxa<-fossilRecord2fossilTaxa(record)
sum(taxa[,5])                 #number of still-living lineages (inc. cryptic)
length(unique(taxa[taxa[,5]==1,6]))     #number of still-living morph-dist. taxa

# and count.cryptic=TRUE
record <- simFossilRecord(p=0.1, q=0.1, r=0.1,
anag.rate=0, prop.bifurc=0, prop.cryptic=0.5,
nruns=1, nExtant=10, totalTime=c(1,100), count.cryptic=TRUE)
taxa<-fossilRecord2fossilTaxa(record)
sum(taxa[,5])                 #number of still-living lineages (inc. cryptic)
length(unique(taxa[taxa[,5]==1,6]))     #number of still-living morph-dist. taxa

##################################################

# an example using startTaxa to have more initial taxa
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
nTotalTaxa=100, startTaxa=20, plot=TRUE)

#######################################################

# Using run conditions

# Users can generate datasets that meet multiple conditions:
# such as time, number of total taxa, extant taxa, sampled taxa
# These can be set as point conditions or ranges

# let's set time = 10-100 units, total taxa = 30-40, extant = 10
#and look at acceptance rates with print.run
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
totalTime=c(10,100), nTotalTaxa=c(30,40), nExtant=10,
print.runs=TRUE, plot=TRUE)

# let's make the constraints on totaltaxa a little tighter
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
totalTime=c(50,100), nTotalTaxa=30, nExtant=10,
print.runs=TRUE, plot=TRUE)
# still okay acceptance rates

# alright, now let's add a constraint on sampled taxa
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
totalTime=c(50,100), nTotalTaxa=30, nExtant=10,
nSamp=15, print.runs=TRUE, plot=TRUE)
# still okay acceptance rates

# we can be really odd and condition on having a single taxon
set.seed(1)
record <- simFossilRecord(p=0.1,
q=0.1, r=0.1, nTotalTaxa=1,
totalTime=c(10,20), plot=TRUE)

#########################################################

# Simulations of entirely extinct taxa

#Typically, a user may want to condition on a precise
# number of sampled taxa in an all-extinct simulation
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
nTotalTaxa=c(1,100), nExtant=0, nSamp=20,
print.runs=TRUE, plot=TRUE)

# Note that when simulations don't include
# sampling or extant taxa, the plot
# functionality changes
record <- simFossilRecord(p=0.1, q=0.1, r=0, nruns=1,
nExtant=0, print.runs=TRUE, plot=TRUE)
# something similar happens when there is no sampling
# and there are extant taxa but they aren't sampled
record <- simFossilRecord(p=0.1, q=0.1, r=0, nruns=1,
nExtant=10, nTotalTaxa=100, modern.samp.prob=0,
print.runs=TRUE, plot=TRUE)


# We can set up a test to make sure that no extant taxa somehow get
# returned in many simulations with extinct-only conditioning:
res<-simFossilRecord(p=0.1, q=0.1, r=0.1,nTotalTaxa=10,nExtant=0,nruns=1000,plot=TRUE)
anyLive<-any(sapply(res,function(z) any(sapply(z,function(x) x[[1]][5]==1))))
if(anyLive){
stop("Runs have extant taxa under conditioning for none?")
}
```

`simFossilRecordMethods` *Methods for Editing or Converting Output from simFossilRecord*

Description

These are a set of functions available for manipulating, translating and editing the objects of class `fossilRecordSimulation` output from function `simFossilRecord`.

Usage

```
timeSliceFossilRecord(fossilRecord, sliceTime, shiftRoot4TimeSlice = FALSE,
  modern.samp.prob = 1, tolerance = 10^-4)

fossilRecord2fossilTaxa(fossilRecord)

fossilRecord2fossilRanges(fossilRecord, merge.cryptic = TRUE,
  ranges.only = TRUE)
```

Arguments

`fossilRecord`	A list object output by `simFossilRecord`, often composed of multiple elements, each of which is data for 'one taxon', with the first element being a distinctive six-element vector composed of numbers, corresponding to the six fields in tables output by the deprecated function `simFossilTaxa`.
`sliceTime`	The date to slice the `simFossilRecord` output at, given in time-units before the modern, on the same scale as the input `fossilRecord`.
`shiftRoot4TimeSlice`	Should the dating of events be shifted, so that the date given for `sliceTime` is now 0, or should the dates not be shifted, so that they remain on the same scale as the input? This argument accepts a logical TRUE or FALSE, but also accepts the string `"withExtantOnly"`, which will only 'shift' the time-scale if living taxa are present, as determined by having ranges that overlap within `tolerance` of `sliceTime`.
`modern.samp.prob`	The probability that a taxon is sampled at the modern time (or, for `timeSliceFossilRecord`, the time at which the simulation data is slice). Must be a number between 0 and 1. If 1, all taxa that survive to the modern day (to the `sliceTime`) are sampled, if 0, none are.
`tolerance`	A small number which sets a range around the `sliceTime` within which taxa will be considered extant for the purposes of output.
`merge.cryptic`	If `TRUE`, sampling events for cryptic taxon-units (i.e. those in the same cryptic complex) will be merged into sampling events for a single taxon-unit (with the name of the first taxon in that cryptic complex).
`ranges.only`	If `TRUE` (the default), `fossilRecord2fossilRanges` will return the dates of the first and last sampled occurrences of each taxon-unit (i.e. the stratigraphic range of each taxon). If `FALSE`, instead a list will be output, with each element being a vector of dates for all sampling events of each taxon-unit.

Details

These functions exist to manipulate `fossilRecordSimulation` objects output from `simFossilRecord`, particularly so that they can be interfaced with functions in library `paleotree` in the same way that output from the deprecated 'legacy' simulation function `simFossilTaxa` was used.

`timeSliceFossilRecord` takes a given `fossilRecordSimulation` object and 'slices' the data to remove any events that occur after the given `sliceTime` and make it so any taxa still alive as of `sliceTime` are now listed as extant.

`fossilRecord2fossilTaxa` converts a `fossilRecordSimulation` object to the flat table format of taxon data as was originally output by deprecated function `simFossilTaxa`, and can be taken as input by a number of `paleotree` functions such as `sampleRanges`,`taxa2phylo` and `taxa2cladogram`.

`fossilRecord2fossilRanges` converts a `fossilRecordSimulation` object to the flat table format of observed taxon ranges, as is typically output by processing `simFossilRecord` simulation output with `paleotree` function `sampleRanges`.

Value

Depends on the function and the arguments given. See Details.

Author(s)

David W. Bapst.

See Also

`simFossilRecord`

Examples

```
set.seed(44)
record <- simFossilRecord(p=0.1, q=0.1, r=0.1, nruns=1,
nTotalTaxa=c(20,30) ,nExtant=0, plot=TRUE)

# time-slicing

# let's try slicing this record at 940 time-units
slicedRecord<-timeSliceFossilRecord(fossilRecord = record, sliceTime = 940)
# and let's plot it
divCurveFossilRecordSim(slicedRecord)

# now with shiftRoot4TimeSlice = TRUE to shift the root age
slicedRecord<-timeSliceFossilRecord(fossilRecord = record, sliceTime = 940,
shiftRoot4TimeSlice = TRUE)
# and let's plot it
divCurveFossilRecordSim(slicedRecord)

# plot look a little different due to how axis limits are treated...
# notice that in both, 'modern' (extant) taxa are sampled with probability = 1
#let's try it again, make that probability = 0

# now with shiftRoot4TimeSlice=TRUE
slicedRecord<-timeSliceFossilRecord(fossilRecord = record, sliceTime = 940,
shiftRoot4TimeSlice = TRUE, modern.samp.prob = 0)
# and let's plot it
divCurveFossilRecordSim(slicedRecord)

############################

# converting to taxa objects and observed ranges
# convert to taxa data
taxa<-fossilRecord2fossilTaxa(record)
# convert to ranges
ranges<-fossilRecord2fossilRanges(record)

# plot diversity curves with multiDiv
multiDiv(list(taxa,ranges),plotMultCurves=TRUE)
# should look a lot like what we got earlier

# get the cladogram we'd obtain for these taxa with taxa2cladogram
cladogram<-taxa2cladogram(taxa,plot=TRUE)

# now get the time-scaled phylogenies with taxa2phylo

# first, with tips extending to the true times of extinction
treeExt<-taxa2phylo(taxa,plot=TRUE)

# now, with tips extending to the first appearance dates (FADs) of taxa
# get the FADs from the ranges
FADs<-ranges[,1]
treeFAD<-taxa2phylo(taxa,FADs,plot=TRUE)
```

`SongZhangDicrano` *Cladistic Data for Dicranograptid Graptolites from Song and Zhang (2014)*

Description

Character matrix and majority-rule cladogram for 12 dicranograptid (and outgroup) graptoloids, taken from Song and Zhang (2014). Included here for use with functions related to character change.

Format

Loading this dataset adds two objects to the R environment. `charMatDicrano` is a `data.frame` object composed of multiple factors, with `NA` values representing missing values (states coded as '?'), read in with `readNexus` from package `phylobase`. `cladogramDicrano` is a cladogram, formatted as a `phylo` class object for use with package `ape`, without branch-lengths (as this was a consensus tree from a maximum-parsimony analysis).

Details

This example dataset is composed of a small cladistic character data for 12 taxa and 24 characters, taken from Song and Zhang (2014). Note that character 22 is a biostratigraphic character, which was not included in all analyses by Song and Zhang.

The included cladogram is the majority-rule consensus of a maximum-parsimony analysis on 12 taxa with 24 characters, including a biostratigraphic character. This tree is included (among the four depicted) as it appeared to be the basis for the majority of Song and Zhang's discussion of dicranograptid systematics.

Both the matrix and the tree were entered by hand from their flat graphic depiction in Song and Zhang's manuscript.

Source

Song, Y., and Y. Zhang. 2014. A preliminary study on the relationship of the early dicranograptids based on cladistic analysis. *GFF* 136(1):243-248.

Examples

```
## Not run:



require(ape)
require(phylobase)

charMatDicrano<-readNexus(file.choose(),type="data",SYMBOLS = " 0 1 2")

cladogramDicrano<-read.nexus(file.choose())

save(charMatDicrano,cladogramDicrano,file="SongZhangDicrano.rdata")


## End(Not run)

data(SongZhangDicrano)

# calculate a distance matrix from the character data
char<-charMatDicrano
charDist<-matrix(,nrow(char),nrow(char))
rownames(charDist)<-colnames(charDist)<-rownames(char)
for(i in 1:nrow(char)){for(j in 1:nrow(char)){
charDiff<-logical()
for(k in 1:ncol(char)){
selectPair<-char[c(i,j),k]
if(all(!is.na(selectPair))){
#drop states that are missing
isSame<-identical(selectPair[1],selectPair[2])
charDiff<-c(charDiff,isSame)
}
}
charDist[i,j]<-1-sum(charDiff)/length(charDiff)
}}

#####
# the following is mostly stolen from the example code for my graptDisparity dataset

#now, is the diagonal zero? (it should be)
```

```
all(diag(charDist)==0)

#now, is the matrix symmetric? (it should be)
isSymmetric(charDist)

#can apply cluster analysis
clustRes <- hclust(as.dist(charDist))
plot(clustRes)

#can apply PCO (use lingoes correction to account for negative values
   #resulting from non-euclidean matrix
pco_res <- pcoa(charDist,correction="lingoes")

#relative corrected eigenvalues
rel_corr_eig <- pco_res$values$Rel_corr_eig
layout(1:2)
plot(rel_corr_eig)
#cumulative
plot(cumsum(rel_corr_eig))

#well let's look at those PCO axes anyway
layout(1)
pco_axes <- pco_res$vectors
plot(pco_axes[,1],pco_axes[,2],pch=16,
   xlab=paste("PCO Axis 1, Rel. Corr. Eigenvalue =",round(rel_corr_eig[1],3)),
   ylab=paste("PCO Axis 2, Rel. Corr. Eigenvalue =",round(rel_corr_eig[2],3)))

#######

# plot majority rule tree from Song and Zhang
plot(cladogramDicrano,
main="MajRule_24charX12Taxa_wBiostratChar")
```

`taxa2cladogram` *Convert Simulated Taxon Data into a Cladogram*

Description

Convert ancestor-descendant relationships of taxa into an 'ideal' unscaled cladogram, where taxa that could share true synapomorphies are arranged into nested clades.

Usage

```
taxa2cladogram(taxad, drop.cryptic = FALSE, plot = FALSE)
```

Arguments

`taxad`	A five-column matrix of taxonomic data, as output by `simFossilRecord`, after transformation with function `fossilRecord2fossilTaxa`.
`drop.cryptic`	Should cryptic species be dropped (except for the first)? Not dropped by default.
`plot`	Should the result be plotted?

Details

This function simulates an ideal cladistic process, where the relationships of a set of morphologically static taxa is resolved into a set of nested hierarchial relationships (a standard cladogram), as much as would be expected given the input relationships among those taxa. taxa2cladogram uses information on the ancestor-descendant relationships of a bunch of taxa and constructs an unscaled cladogram of the hierarcially-nesting relationships among those taxa. There's no actual cladistics going on, this is just a simulation of that process. If there is any chance that a set of taxa could be resolved into a set of nested relationships given their ancestor-descendant relationships, they will be resolved so in the output of taxa2cladogram. No morphological characters are considered, we just assume that if there is a nesting relationship, then it could be resolved as such. This makes it the "ideal" cladogram of a simulated clade.

The result will probably not be fully resolved, as including both ancestor and descendant taxa will generally make it impossible to produce a fully nesting system of relationships. For example, consider a set of three morphologically-static taxa where the first is an ancestor (either direct or indirect, ala Foote, 1996) of both the second and third. If we imagine an ideal cladistic analysis of the morphological characters of those three taxa, this set of taxa will be unable to be broken up into bifurcating-nested relationships and thus result in a polytomy. Any set of ancestor-descendant relationships will have many of these, as some ancestors must have more than one descendant for the clade to diversify, as noted by Wagner and Erwin, 1995.

If there are cryptic taxa present in the output from `simFossilRecord`, these and any of their morphologically distinguishable descendants are collapsed into a polytomy to simulate the expected pattern of lack of phylogenetic resolution. In addition to this merging, cryptic taxa can be dropped via the argument drop.cryptic, such that only the first 'species' of each cryptic taxon assemblage is listed among the tip taxa (what we would actually expect to obtain, as wouldn't recognize cryptic taxa as different OTUs). By default, cryptic taxa are not dropped so that the same number of taxa as in the simulated data is retained.

Value

The resulting phylogeny without branch lengths is output as an object of class phylo.

The tip labels are the rownames from the simulation input; see documentation for `simFossilRecord` and `fossilRecord2fossilTaxa` documentation for details.

Author(s)

David W. Bapst

References

Foote, M. 1996 On the Probability of Ancestors in the Fossil Record. *Paleobiology* **22**(2):141-151.

Wagner, P., and D. Erwin. 1995 Phylogenetic patterns as tests of speciation models. New approaches to speciation in the fossil record. Columbia University Press, New York:87-122.

See Also

simFossilRecord, taxa2phylo, fossilRecord2fossilTaxa

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
layout(1:2)
cladogram<-taxa2cladogram(taxa,plot=TRUE)
#compare the "real" time-scaled tree of taxon last occurrences (taxa2phylo)
     #to the 'ideal' cladogram
tree<-taxa2phylo(taxa,plot=TRUE)

#testing with cryptic speciation
recordCrypt<-simFossilRecord(p=0.1, q=0.1, prop.cryptic=0.5, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxaCrypt<-fossilRecord2fossilTaxa(recordCrypt)
layout(1:2)
parOrig<-par(no.readonly=TRUE)
par(mar=c(0,0,0,0))
cladoCrypt1<-taxa2cladogram(taxaCrypt,drop.cryptic=FALSE)
plot(cladoCrypt1)
cladoCrypt2<-taxa2cladogram(taxaCrypt,drop.cryptic=TRUE)
plot(cladoCrypt2)

#reset plotting
par(parOrig)
layout(1)
```

---

taxa2phylo *Convert Simulated Taxon Data into a Phylogeny*

Description

Converts temporal and ancestor-descendant relationships of taxa into a time-scaled phylogeny

Usage

```
taxa2phylo(taxad, obs_time = NULL, plot = FALSE)
```

Arguments

`taxad`	A five-column matrix of taxonomic data, as output by `fossilRecord2fossilTaxa` via simulations produced using `simFossilRecord`
`obs_time`	A vector of per-taxon times of observation which must be in the same order of taxa as in the object taxad; if NULL, the LADs (column 4) in taxad2 are used
`plot`	Plot the resulting phylogeny?

Details

As described in the documentation for taxa2cladogram, the relationships among morphotaxa in the fossil record are difficult to describe in terms of traditional phylogenies. One possibility is to arbitrarily choose particular instantaneous points of time in the range of some taxa and describe the time-scaled relationships of the populations present at those dates. This is the tactic used by taxa2phylo.

By default, the dates selected ('obs-time' argument) are the last occurances of the taxon, so a simple use of this function will produce a time-scaled tree which describes the relaitonships of the populations present at the last occurance of each taxon in the sampled data. Alternatively, obs_time can be supplied with different dates within the taxon ranges.

All data relating to when static morpho-taxa appear or disappear in the record is lost; branching points will be the actual time of speciation, which (under budding) will often be in the middle of the temporal range of a taxon.

Cryptic taxa are not dropped or merged as can be done with taxa2cladogram. The purpose of taxa2phylo is to obtain the 'true' pattern of evolution for the observation times, independent of what we might actually be able to recover, for the purpose of comparing in simulation analyses.

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

The resulting phylogeny with branch lengths is output as an object of class phylo. This function will output trees with the element $root.time, which is the time of the root divergence in absolute time.

The tip labels are the rownames from the simulation input; see documentation for `simFossilRecord` and `fossilRecord2fossilTaxa` documentation for details.

Note

DO NOT use this function to time-scale a real tree for a real dataset. It assumes you know the divergence/speciation times of the branching nodes and relationships perfectly, which is almost impossible given the undersampled nature of the fossil record. Use timePaleoPhy or cal3TimePaleoPhy instead.

DO use this function when doing simulations and you want to make a tree of the 'true' history, such as for simulating trait evolution along phylogenetic branches.

Unlike `taxa2cladogram`, this function does not merge cryptic taxa in output from `simFossilRecord` (via `fossilRecord2fossilTaxa`) and I do not offer an option to secondarily drop them. The tip labels should provide the necessary information for users to drop such taxa, however. See simFossilRecord.

Author(s)

David W. Bapst

See Also

simFossilRecord, taxa2cladogram, fossilRecord2fossilTaxa

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
  nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
tree<-taxa2phylo(taxa)
phyloDiv(tree)

#now a phylogeny with tips placed at the apparent time of extinction for each taxon
rangesCont<-sampleRanges(taxa,r=0.5)
tree<-taxa2phylo(taxa,obs_time=rangesCont[,2])
phyloDiv(tree,drop.ZLB=FALSE)
#note that it drops taxa which were never sampled!

#testing with cryptic speciation
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, prop.cryptic=0.5, nruns=1,
  nTotalTaxa=c(30,40), nExtant=0, count.cryptic=TRUE)
taxaCrypt<-fossilRecord2fossilTaxa(record)
treeCrypt<-taxa2phylo(taxaCrypt)
layout(1)
plot(treeCrypt)
axisPhylo()
```

taxonSortPBDBocc *Sorting Unique Taxa of a Given Rank from Paleobiology Database Occurrence Data*

Description

Functions for sorting out unique taxa from Paleobiology Database occurrence downloads, which should accept several different formats resulting from different versions of the PBDB API and different vocabularies available from the API.

Usage

```
taxonSortPBDBocc(data, rank, onlyFormal = TRUE, cleanUncertain = TRUE,
  cleanResoValues = c(NA, "\"", "", "n. sp.", "n. gen.", " ", "  "))
```

Arguments

`data`	A table of occurrence data collected from the Paleobiology Database.
`rank`	The selected taxon rank; must be one of 'species', 'genus', 'family', 'order', 'class' or 'phylum'.
`onlyFormal`	If TRUE (the default) only taxa formally accepted by the Paleobiology Database are returned. If FALSE, then the identified name fields are searched for any additional 'informal' taxa with the proper taxon. If their taxon name happens to match any formal taxa, their occurrences are merged onto the formal taxa. This argument generally has any appreciable effect when rank=species.
`cleanUncertain`	If TRUE (the default) any occurrences with an entry in the respective 'resolution' field that is *not* found in the argument cleanResoValue will be removed from the dataset. These are assumed to be values indicating taxonomic uncertainty, i.e. 'cf.' or '?'.
`cleanResoValues`	The set of values that can be found in a 'resolution' field that do not cause a taxon to be removed, as they do not seem to indicate taxonomic uncertainty.

Details

Data input for `taxonSortPBDBocc` are expected to be from version 1.2 API with the 'pbdb' vocabulary. However, datasets are passed to internal function `translatePBDBocc`, which attempts to correct any necessary field names and field contents used by `taxonSortPBDBocc`.

This function can pull either *just* the 'formally' identified and synonymized taxa in a given table of occurrence data or pull *in addition* occurrences listed under informal taxa of the sought taxonomic rank. Only formal taxa are sorted by default; this is controlled by argument `onlyFormal`. Pulling the informally-listed taxonomic occurrences is often necessary in some groups that have received little focused taxonomic effort, such that many species are linked to their generic taxon ID and never received a species-level taxonomic ID in the PBDB. Pulling both formal and informally listed taxonomic occurrences is a hierarchical process and performed in stages: formal taxa are identified first, informal taxa are identified from the occurrences that are 'leftover', and informal occurrences with name labels that match a previously sorted formally listed taxon are concatenated to the 'formal' occurrences for that same taxon, rather than being listed under separate elements of the list as if they were separate taxa. This function is simpler than similar functions that inspired it by using the input"rank" to both filter occurrences and directly reference a taxon's accepted taxonomic placement, rather than a series of specific `if()` checks.

Unlike some similar functions in other packages, such as version 0.3 `paleobioDB`'s `pbdb_temp_range`, `taxonSortPBDBocc` does not check if sorted taxa have a single 'taxon_no' ID number. This makes the blanket assumption that if a taxon's listed name in relevant fields is identical, the taxon is identical, with the important caveat that occurrences with accepted formal synonymies are sorted first based on their accepted names, followed by taxa without formal taxon IDs. This should avoid mistakingly linking the same occurrences to multiple taxa or assigning occurrences listed under separate formal taxa to the same taxon based on their 'identified' taxon name, as long as all formal taxa have unique names (which is an untested assumption). In some cases, this procedure is helpful, such as when taxa with identical generic and species names are listed under separate taxon ID numbers because of a difference in the listed subgenus for some occurrences (example, "Pseudoclimacograptus (Metaclimacograptus) hughesi' and 'Pseudoclimacograptus hughesi' in the PBDB as of 03/01/2015). Presumably any data that would be affected by differences in this procedure is very minor.

Occurrences with taxonomic uncertainty indicators in the listed identified taxon name are removed by default, as controlled by argument `cleanUncertain`. This is done by removing any occurrences that have an entry in `primary_reso` (was "genus_reso" in v1.1 API) when `rank` is a supraspecific level, and `species_reso` when rank=species, if that entry is not found in `cleanResoValues`. In some rare cases, when `onlyFormal=FALSE`, supraspecific taxon names may be returned in the output that have various 'cruft' attached, like 'n.sp'.

Empty values in the input data table ("") are converted to NAs, as they may be due to issues with using read.csv to convert API-downloaded data.

Value

Returns a list where each element is different unique taxon obtained by the sorting function, and named with that taxon name. Each element is composed of a table containing all the same occurrence data fields as the input (potentially with some fields renamed and some field contents change, due to vocabulary translation).

Author(s)

David W. Bapst, but partly inspired by Matthew Clapham's `cleanTaxon` (found at this location on github) and R package paleobioDB's `pbdb_temp_range` function (found at this location on github.

See Also

`occData2timeList`, `plotOccData` and the example graptolite dataset at `graptPBDB`

Examples

```
#load example graptolite PBDB occ dataset
data(graptPBDB)

#get formal genera
occGenus<-taxonSortPBDBocc(graptOccPBDB, rank="genus")
length(occGenus)

#get formal species
occSpeciesFormal<-taxonSortPBDBocc(graptOccPBDB, rank="species")
length(occSpeciesFormal)

#yes, there are fewer 'formal' graptolite species in the PBDB then genera

#get formal and informal species
occSpeciesInformal<-taxonSortPBDBocc(graptOccPBDB, rank="species",
 onlyFormal=FALSE)
length(occSpeciesInformal)

#way more graptolite species are 'informal' in the PBDB

#get formal and informal species
#including from occurrences with uncertain taxonomy
```

```
#basically everything and the kitchen sink
occSpeciesEverything<-taxonSortPBDBocc(graptOccPBDB, rank="species",
onlyFormal=FALSE, cleanUncertain=FALSE)
length(occSpeciesEverything)

## Not run:

# simple function for getting occurrence data from API v1.1
easyGetPBDBocc<-function(taxa,show=c("ident","phylo")){
  #cleans PBDB occurrence downloads of warnings
  taxa<-paste(taxa,collapse=",")
taxa<-paste(unlist(strsplit(taxa,"_")),collapse="%20")
show<-paste(show,collapse=",")
command<-paste0("http://paleobiodb.org/data1.2/occs/list.txt?base_name=",
taxa,"&show=",show,"&limit=all",
collapse="")
command<-paste(unlist(strsplit(command,split=" ")),collapse="%20")
downData<-readLines(command)
if(length(grep("Warning",downData))!=0){
start<-grep("Records",downData)
warn<-downData[1:(start-1)]
warn<-sapply(warn, function(x)
paste0(unlist(strsplit(unlist(strsplit(x,'"')),",")),collapse=""))
warn<-paste0(warn,collapse="\n")
names(warn)<-NULL
mat<-downData[-(1:start)]
mat<-read.csv(textConnection(mat))
message(warn)
}else{
mat<-downData
mat<-read.csv(textConnection(mat))
}
return(mat)
}

#try a PBDB API download with lots of synonymization
#this should have only 1 species
#old way:
#acoData<-read.csv(paste0("http://paleobiodb.org/data1.1/occs/list.txt?",
# "base_name=Acosarina%20minuta&show=ident,phylo&limit=all"))
# with easyGetPBDBocc:
acoData<-easyGetPBDBocc("Acosarina minuta")
x<-taxonSortPBDBocc(acoData, rank="species", onlyFormal=FALSE)
names(x)

#make sure works with API v1.1
dicelloData<-read.csv(paste0("http://paleobiodb.org",
"/data1.1/occs/list.txt?base_name=Dicellograptus",
"&show=ident,phylo&limit=all"))
dicelloOcc2<-taxonSortPBDBocc(dicelloData, rank="species", onlyFormal=FALSE)
names(dicelloOcc2)

#make sure works with compact vocab v1.1
```

```
dicelloData<-read.csv(paste0("http://paleobiodb.org",
"/data1.1/occs/list.txt?base_name=Dicellograptus",
"&show=ident,phylo&limit=all&vocab=com"))
dicelloOccCom1<-taxonSortPBDBocc(dicelloData, rank="species", onlyFormal=FALSE)
names(dicelloOccCom1)
head(dicelloOccCom1[[1]])[,1:7]

#make sure works with compact vocab v1.2
dicelloData<-read.csv(paste0("http://paleobiodb.org",
"/data1.2/occs/list.txt?base_name=Dicellograptus",
"&show=ident,phylo&limit=all&vocab=com"))
dicelloOccCom1<-taxonSortPBDBocc(dicelloData, rank="species", onlyFormal=FALSE)
names(dicelloOccCom1)
head(dicelloOccCom1[[1]])[,1:7]


## End(Not run)
```

---

taxonTable2taxonTree *Create a Taxonomy-Based Phylogeny ('Taxon Tree') from a Hierarchical Table of Taxonomy Memberships*

Description

This function takes a matrix of taxon names, indicating a set of hierarchical taxonomic relationships conveyed as nested placements for a set of tip-taxa (listed in the last column of the matrix) and returns a 'taxonomy-tree' phylogeny object of class 'phylo'.

Usage

```
taxonTable2taxonTree(taxonTable, cleanTree = TRUE)
```

Arguments

`taxonTable`	A matrix of type character and multiple rows and columns, containing the tip taxa in the last column, one per row, with progressively larger taxa listed in prior columns (reading left-to-right). Invariant columns (i.e. taxa that all tip taxa are in) are allowed, but all but the most 'shallow' of such invariant taxa are dropped prior to transformation to a taxon-tree phylogeny object.
`cleanTree`	By default, the tree is run through a series of post-processing, including having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions. If FALSE, none of this post-processing is done and users should beware, as such trees can lead to hard-crashes of R.

Details

This function can deal with empty entries in cells of `taxonTable` by assuming these are lower-level taxa which are 'floating' freely somewhere in taxa several levels higher.

Value

A phylogeny of class 'phylo', where each tip is a taxon listed in the last column of the input taxonTable. Edges are scaled so that the distance from one taxon rank to another 1, then merged to remove singleton nodes. As not all taxa have parents at the immediate taxon level above, this leads to some odd cases. For example, two genera emanating from a node representing a class but with a very short (length=1) branch and a long branch (length=3) means one genus is simply placed in the class, with no family or order listed while the one on the long branch is within an order and family that is otherwise monogeneric.

The names of higher taxa than the tips should be appended as the element $node.label for the internal nodes.

Author(s)

David W. Bapst

See Also

makePBDBtaxonTree, parentChild2taxonTree

Examples

```
#let's create a small, really cheesy example
pokeTable<-rbind(cbind("Pokezooa","Shelloidea","Squirtadae",
c("Squirtle","Blastoise","Wartortle")),
c("Pokezooa","Shelloidea","","Lapras"),
c("Pokezooa","","","Parasect"),
cbind("Pokezooa","Hirsutamona","Rodentapokemorpha",
c("Linoone","Sandshrew","Pikachu")),
c("Pokezooa","Hirsutamona",NA,"Ursaring"))

pokeTree<-taxonTable2taxonTree(pokeTable)
plot(pokeTree);nodelabels(pokeTree$node.label)
```

---

termTaxa *Simulating Extinct Clades of Monophyletic Taxa*

Description

This function simulates the diversification of clades composed of monophyletic terminal taxa, which are distinguished in a fashion completely alternative to way taxa are defined in the simulation functions `simFossilRecord`, `taxa2cladogram` and `taxa2phylo`.

Usage

```
simTermTaxa(ntaxa, sumRate = 0.2)

simTermTaxaAdvanced(p = 0.1, q = 0.1, mintaxa = 1, maxtaxa = 1000,
  mintime = 1, maxtime = 1000, minExtant = 0, maxExtant = NULL,
  min.cond = TRUE)

trueTermTaxaTree(TermTaxaRes, time.obs)

deadTree(ntaxa, sumRate = 0.2)
```

Arguments

`ntaxa`	Number of monophyletic 'terminal' taxa (tip terminals) to be included on the simulated tree
`sumRate`	The sum of the instantaneous branching and extinction rates; see below.
`p`	Instantaneous rate of speciation/branching.
`q`	Instantaneous rate of extinction.
`mintaxa`	Minimum number of total taxa over the entire history of a clade necessary for a dataset to be accepted.
`maxtaxa`	Maximum number of total taxa over the entire history of a clade necessary for a dataset to be accepted.
`mintime`	Minimum time units to run any given simulation before stopping.
`maxtime`	Maximum time units to run any given simulation before stopping.
`minExtant`	Minimum number of living taxa allowed at end of simulations.
`maxExtant`	Maximum number of living taxa allowed at end of simulations.
`min.cond`	If TRUE, the default, simulations are stopped when they meet all minimum conditions. If FALSE, simulations will continue until they hit maximum conditions, but are only accepted as long as they still meet all minimum conditions in addition.
`TermTaxaRes`	The list output produced by simTermTaxa
`time.obs`	A per-taxon vector of times of observation for the taxa in TermTaxaRes

Details

`deadTree` generates a time-scaled topology for an entirely extinct clade of a specific number of tip taxa. Because the clade is extinct and assumed to have gone extinct in the distant past, many details of typical birth-death simulators can be ignored. If a generated clade is already conditioned upon the (a) that some number of taxa was reached and (b) then the clade went extinct, the topology (i.e. the distribution of branching and extinction events) among the branches should be independent of the actual generating rate. The frequency of nodes is a simple mathematical function of the number of taxa (i.e. number of nodes is the number of taxa -1) and their placement should completely random, given that we generally treat birth-death processes as independent Poisson processes. Thus, in terms of generating the topology, this function is nothing but a simple wrapper for the ape function rtree, which randomly places splits among a set of taxa using a simple algorithm (see Paradis, 2012). To match the expectation of a birth-death process, new branch lengths are calculated as an exponential distribution with mean 1/sumRate, where sumRate represents the sum of the branching and extinction rates. Although as long as both the branching rate and extinction rates are more than zero, any non-ultrametric tree is possible, only when the two rates are non-zer and equal to each other will there be a high chance of getting an extinct clade with many tips. Any analyses one could do on a tree such as this will almost certainly give estimates of equal branching and extinction rates, just because all taxa are extinct.

simTermTaxa produces 'terminal-taxon' datasets; datasets of clades where the set of distinguishable taxa are defined as intrinsically monophyletic. (In version 1.6, I referred to this as the 'candle' mode, so named from the 'candling' horticultural practice and the visual conceptualization of the model.) On theoretical terms, terminal-taxa datasets are what would occur if (a) only descendant lineages can be sample and (b) all taxa are immediately differentiated as of the last speciation event and continue to be so differentiated until they go extinct. In practice, this means the taxa on such a tree would represent a sample of all the terminal branches, which start with some speciation event and end in an extinction event. These are taken to be the true original ranges of these taxa. No further taxa can be sampled than this set, whatsoever. Note that the differentiation here is a result of a posteriori consideration of the phylogeny: one can't even know what lineages could be sampled or the actual start points of such taxa until after the entire phylogeny of a group of organisms is generated.

Because all evolutionary history prior to any branching events is unsampled, this model is somewhat agnostic about the general model of differentiation among lineages. The only thing that can be said is that synapomorphies are assumed to be potentially present along every single branch, such that in an ideal scenario every clade could be defined. This would suggest very high anagenesis or bifurcation.

Because the set of observable taxa is a limited subset of the true evolution history, the true taxon ranges are not a faithful reproduction of the true diversity curve. See an example below.

simTermTaxa uses deadTree to make a phylogeny, so the only datasets produced are of extinct clades. simTermTaxaAdvanced is an alternative to simTermTaxa which uses simFossilRecord to generate the underlying pattern of evolutionary relationships and not deadTree. The arguments are thus similar to simFossilRecord, with some differences (as simTermTaxaAdvanced originally called the deprecated function simFossilTaxa). In particular, simTermTaxaAdvanced can be used to produce simulated datasets which have extant taxa.

trueTermTaxaTree is analagous to the function of taxa2phylo, in that it outputs the time-scaled-phylogeny for a terminal-taxon dataset for some times of observations. Unlike with the use of taxa2phylo on the output on simFossilRecord (via fossilRecord2fossilTaxa, there is no need to use trueTermTaxaTree to obtain the true phylogeny when times of extinction are the times of observation; just get the $tree element from the result output by simTermTaxa.

Also unlike with taxa2phylo, the cladistic topology of relationships among morphotaxa never changes as a function of time of observation. For obtaining the 'ideal cladogram' of relationships among the terminal taxa, merely take the $tree element of the output from simtermTaxaData and remove the branch lengths (see below for an example).

As with many functions in the paleotree library, absolute time is always decreasing, i.e. the present day is zero.

Value

deadTree gives time-scaled phylo object, with a $root.time element. As discussed above, the result is always an extinct phylogeny of exactly `ntaxa`.

`simTermTaxa` and `simTermTaxaAdvanced` both produce a list with two components: `$taxonRanges` which is a two-column matrix where each row gives the true first and last appearance of observable taxa and `$tree` which is a time-scaled phylogeny with end-points at the true last appearance time of taxa.

`trueTermTaxaTree` produces a time-scaled tree as a phylo object, which describes the relationships of populations at the times of observation given in the time.obs argument.

Author(s)

David W. Bapst

References

Paradis, E. (2012) *Analysis of Phylogenetics and Evolution with R (Second Edition).* New York: Springer.

See Also

deadtree is simply a wraper of the function `rtree` in ape.

For a very different way of simulating diversification in the fossil record, see `simFossilRecord`, `fossilRecord2fossilTaxa`, `taxa2phylo` and `taxa2cladogram`.

Examples

```
set.seed(444)
#example for 20 taxa
termTaxaRes<-simTermTaxa(20)

#let look at the taxa...
taxa<-termTaxaRes$taxonRanges
taxicDivCont(taxa)
#because ancestors don't even exist as taxa
#the true diversity curve can go to zero
#kinda bizarre!

#the tree should give a better idea
tree<-termTaxaRes$tree
phyloDiv(tree)
#well, okay, its a tree.

#get the 'ideal cladogram' ala taxa2cladogram
    #much easier with terminal-taxa simulations as no paraphyletic taxa
cladogram<-tree
cladogram$edge.length<-NULL
plot(cladogram)

#trying out trueTermTaxaTree
#random times of observation: uniform distribution
time.obs<-apply(taxa,1,function(x) runif(1,x[2],x[1]))
```

```
tree1<-trueTermTaxaTree(termTaxaRes,time.obs)
layout(1:2)
plot(tree)
plot(tree1)

layout(1)

#let's look at the change in the terminal branches
plot(tree$edge.length,tree1$edge.length)
#can see some edges are shorter on the new tree, cool

#let's now simulate sampling and use FADs
layout(1:2)
plot(tree);axisPhylo()
FADs<-sampleRanges(termTaxaRes$taxonRanges,r=0.1)[,1]
tree1<-trueTermTaxaTree(termTaxaRes,FADs)
plot(tree1);axisPhylo()

#can condition on sampling some average number of taxa
#analagous to deprecated function simFossilTaxa_SRcond
r<-0.1
avgtaxa<-50
sumRate<-0.2
#avg number necc for an avg number sampled
ntaxa_orig<-avgtaxa/(r/(r+sumRate))
termTaxaRes<-simTermTaxa(ntaxa=ntaxa_orig,sumRate=sumRate)
#note that conditioning must be conducted using full sumRate
#this is because durations are functions of both rates
#just like in bifurcation

#use advanced version of simTermTaxa: simTermTaxaAdvanced
    #allows for extant taxa in a term-taxa simulation

#with min.cond
termTaxaRes<-simTermTaxaAdvanced(p=0.1,q=0.1,mintaxa=50,
    maxtaxa=100,maxtime=100,minExtant=10,maxExtant=20,min.cond=TRUE)
#notice that arguments are similar to simFossilRecord
# and somewhat more similar to deprecated function simFossilTaxa ;P
plot(termTaxaRes$tree)
Ntip(termTaxaRes$tree)

#without min.cond
termTaxaRes<-simTermTaxaAdvanced(p=0.1,q=0.1,mintaxa=50,
    maxtaxa=100,maxtime=100,minExtant=10,maxExtant=20,min.cond=FALSE)
plot(termTaxaRes$tree)
Ntip(termTaxaRes$tree)

layout(1)
```

testEdgeMat *Test the Edge Matrix of a 'phylo' Phylogeny Object for Inconsistencies*

Description

`testEdgeMat` is a small simple function which tests the $edge matrix of 'phylo' objects for inconsistencies that can cause downstream analytical problems. The associated function, `cleanNewPhylo` puts an input phylo object, presumably freshly created or reconstituted by some function, through a series of post-processing, This includes having singles collapsed, nodes reordered and being written out as a Newick string and read back in, to ensure functionality with ape functions and ape-derived functions.

Usage

```
testEdgeMat(tree)

cleanNewPhylo(tree)
```

Arguments

`tree`	A phylogeny object of type phylo

Details

Useful when doing complex manipulations and reconstitutions of 'phylo' objects (or their de novo construction), and thus is used by a number of paleotree functions.

Value

For `testEdgeMat`, if all the checks in the function pass correctly, the logical TRUE is returned.

For `cleanNewPhylo`, an object of class 'phylo' is returned.

Author(s)

David W. Bapst, with a large number of tests incorporated from Emmanuel Paradis's checkValid-Phylo function, provided at his github repo here, which was released GPL v>2:

https://github.com/emmanuelparadis/checkValidPhylo

Examples

```
set.seed(444)
tree<-rtree(10)
# should return TRUE
testEdgeMat(tree)

# should also work on star trees
testEdgeMat(stree(10))

# should also work on trees with two taxa
testEdgeMat(rtree(2))

# should also work on trees with one taxon
testEdgeMat(stree(1))
```

```
#running cleanNewPhylo on this tree should have little effect
#beyond ladderizing it...
tree1<-cleanNewPhylo(tree)

#compare outputs
layout(1:2)
plot(tree)
plot(tree1)
layout(1)
```

---

timeLadderTree *Resolve Polytomies by Order of First Appearance*

Description

Resolves polytomies in trees with lineages arranged in a pectinate pattern (i.e. a ladder-like subtree), ordered by the time of first appearance (FAD) for each lineage.

Usage

```
timeLadderTree(tree, timeData)
```

Arguments

`tree`	A phylo object
`timeData`	Two-column matrix of per-taxon first and last occurrances in absolute continous time

Details

This method of resolving polytomies assumes that the order of stratigraphic appearance perfectly depicts the order of branching. This may not be a good assumption for poorly sampled fossil records.

This function is for resolving trees when a continuous time-scale is known. For discrete time-scales, see the function bin_timePaleoPhy.

Taxa with the same identical first appearance date will be ordered randomly. Thus, the output is slightly stochastic, but only when ties exist. This is probably uncommon with real data on continuous time-scales.

Taxa not shared between the tree and the timeData matrix, or listed as having a FAD or LAD of NA in timeData will be dropped and will not be included in the output tree.

See this blog post for more information:

http://nemagraptus.blogspot.com/2012/07/resolving-polytomies-according-to.html

Value

Returns the modified tree as an object of class phylo, with no edge lengths.

Author(s)

David W. Bapst

See Also

di2multi

Examples

```
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(100,200))
taxa<-fossilRecord2fossilTaxa(record)
tree<-taxa2cladogram(taxa)
ranges<-sampleRanges(taxa,r=0.5)
tree1<-timeLadderTree(tree,ranges)
layout(1:2)
plot(ladderize(tree),show.tip.label=FALSE)
plot(ladderize(tree1),show.tip.label=FALSE)

#an example with applying timeLadderTree to discrete time data
rangeData<-binTimeData(ranges,int.len=5) #sim discrete range data
tree2<-bin_timePaleoPhy(tree,timeList=rangeData,timeres=TRUE)
plot(ladderize(tree),show.tip.label=FALSE)
plot(ladderize(tree2),show.tip.label=FALSE)
axisPhylo()

layout(1)
```

`timeList2fourDate` *Converting Datasets of Taxon Ranges in Intervals Between timeList format and fourDate format*

Description

Functions for manipulating data where the first and last appearances of taxa are known from bounded intervals of time. The two main functions listed here are for converting between (1) a data structure consisting of a single 'flat' table where each taxon is listed as a set of four dates (a `fourDate` data type), and (2) a list format where each taxon is listed as its first and last intervals, with an associated table of age bounds for the intervals referred to in the first table (referred to as a `timeList` data structure by many `paleotree` functions).

Usage

```
timeList2fourDate(timeList)

fourDate2timeList(fourDate)
```

Arguments

`timeList`	A list composed of two matrices with two columns each, the first giving interval start and end date bounds, and the second giving taxon first and last interval appearances in reference to the intervals listed in the first matrix.
`fourDate`	A four column matrix where each row is a different taxon, the first two columns are the lower and upper bounds on the time of first appearance for that taxon and the third and fourth columns are respectively the lower and upper bounds on the time of last appearance for that taxon, all in time before present.

Details

`timeList2fourDate` is for converting from a `timeList` format to a `fourDate` format. `fourDate2timeList` is for converting from a `fourDate` format to a `timeList` format.

Value

A converted data object, respective to the function applied.

Author(s)

David W. Bapst

References

See my recent blog post on temporal datasets in paleontology for some details:

http://nemagraptus.blogspot.com/2015/02/how-do-we-treat-fossil-age-data-dates.html

See Also

`bin_timePaleoPhy` and `taxicDivDisc` for common applications; `binTimeData` for a simulation function for such data objects

Examples

```
#timeList object from the retiolinae dataset
data(retiolitinae)

str(retioRanges)

taxicDivDisc(retioRanges)

fourDateRet<-timeList2fourDate(retioRanges)

# total uncertainty in retio first and last appearances?
sum((fourDateRet[,1]-fourDateRet[,2])+(fourDateRet[,3]-fourDateRet[,4]))

#convert back
newTimeList<-fourDate2timeList(fourDateRet)
taxicDivDisc(retioRanges)
```

timePaleoPhy	*Typical 'a posteriori' Time-Scaling Approaches For Paleontological Phylogenies*

Description

Time-scales an unscaled cladogram of fossil taxa using information on their temporal ranges, using various methods. Also can resolve polytomies randomly and output samples of randomly-resolved trees. As simple methods of time-scaling phylogenies of fossil taxa can have biasing effects on macroevolutionary analyses (Bapst, 2014, Paleobiology), this function is largely retained for legacy purposes and plotting applications. The time-scaling methods implemented by the functions listed here do **not** return realistic estimates of divergence dates, users should investigate other time-scaling methods such as `cal3TimePaleoPhy`.

Usage

```
timePaleoPhy(tree, timeData, type = "basic", vartime = NULL, ntrees = 1,
  randres = FALSE, timeres = FALSE, add.term = FALSE,
  inc.term.adj = FALSE, dateTreatment = "firstLast", node.mins = NULL,
  noisyDrop = TRUE, plot = FALSE)

bin_timePaleoPhy(tree, timeList, type = "basic", vartime = NULL,
  ntrees = 1, nonstoch.bin = FALSE, randres = FALSE, timeres = FALSE,
  sites = NULL, point.occur = FALSE, add.term = FALSE,
  inc.term.adj = FALSE, dateTreatment = "firstLast", node.mins = NULL,
  noisyDrop = TRUE, plot = FALSE)
```

Arguments

`tree`	An unscaled cladogram of fossil taxa, of class `phylo`. Tip labels must match the taxon labels in the respective temporal data.
`timeData`	Two-column matrix of first and last occurrences in absolute continuous time, with row names as the taxon IDs used on the tree. This means the first column is very precise FADs (first appearance dates) and the second column is very precise LADs (last appearance dates), reflect the precise points in time when taxa first and last appear. If there is stratigraphic uncertainty in when taxa appear in the fossil record, it is preferable to use the 'bin' time-scaling functions; however, see the argument `dateTreatment`.
`type`	Type of time-scaling method used. Can be "basic", "equal", "equal_paleotree_legacy", "equal_date.phylo_legacy" "aba", "zbla" or "mbl". Type="basic" by default. See details below.
`vartime`	Time variable; usage depends on the method 'type' argument. Ignored if type = "basic".
`ntrees`	Number of time-scaled trees to output. If ntrees is greater than one and both randres is false and dateTreatment is neither 'minMax' or 'randObs', the function will fail and a warning is issued, as these arguments would simply produce multiple identical time-scaled trees.

`randres`	Should polytomies be randomly resolved? By default, `timePaleoPhy` does not resolve polytomies, instead outputting a time-scaled tree that is only as resolved as the input tree. If `randres = T`, then polytomies will be randomly resolved using `multi2di` from the package ape. If `randres = T` and `ntrees = 1`, a warning is printed that users should analyze multiple randomly-resolved trees, rather than a single such tree, although a tree is still output.
`timeres`	Should polytomies be resolved relative to the order of appearance of lineages? By default, `timePaleoPhy` does not resolve polytomies, instead outputting a time-scaled tree that is only as resolved as the input tree. If `timeres = TRUE`, then polytomies will be resolved with respect to time using the paleotree function `timeLadderTree`. See that functions help page for more information; the result of time-order resolving of polytomies generally does not differ across multiple uses, unlike use of `multi2di`.
`add.term`	If true, adds terminal ranges. By default, this function will not add the ranges of taxa when time-scaling a tree, so that the tips correspond temporally to the first appearance datums of the given taxa. If `add.term = T`, then the 'terminal ranges' of the taxa are added to the tips after tree is time-scaled, such that the tips now correspond to the last appearance datums.
`inc.term.adj`	If true, includes terminal ranges in branch length estimates for the various adjustment of branch lengths under all methods except 'basic' (i.e. a terminal length branch will not be treated as zero length is this argument is `TRUE` if the taxon at this tip has a non-zero duration). By default, this argument is `FALSE` and this function will not include the ranges of taxa when adjusting branch lengths, so that zero-length branches before first appearance times will be extended. An error is returned if this argument is true but `type = "basic"` or `add.term = FALSE`, as this argument is inconsistent with those argument options.
`dateTreatment`	This argument controls the interpretation of timeData. The default setting 'firstLast' treats the dates in timeData as a column of precise first and last appearances, such that first appearances will be used to date nodes and last appearances will only be called on if `add.term = TRUE`. A second option, added by great demand, is 'minMax' which treats these dates as minimum and maximum bounds on single point dates. Under this option, all taxa in the analysis will be treated as being point dates, such that the first appearance is also the last. These dates will be pulled under a uniform distribution. If 'minMax' is used, add.term becomes meaningless, and the use of it will return an error message. A third option is 'randObs'. This assumes that the dates in the matrix are first and last appearance times, but that the desired time of observation is unknown. Thus, this is much like 'firstLast' except the effective time of observation (the taxon's LAD under 'firstLast') is treated an uncertain date, and is randomly sampled between the first and last appearance times. The FAD still is treated as a fixed number, used for dating the nodes. In previous versions of paleotree, this was called in `timePaleoPhy` using the argument rand.obs, which has been removed for clarity. This temporal uncertainty in times of observation might be useful if a user is interested in applying phylogeny-based approaches to studying trait evolution, but have per-taxon measurements of traits that come from museum specimens with uncertain temporal placement. With both arguments 'minMax'

	and 'randObs', the sampling of dates from random distributions should compel users to produce many time-scaled trees for any given analytical purpose. Note that 'minMax' returns an error in 'bin' time-scaling functions; please use 'points.occur' instead.
`node.mins`	The minimum dates of internal nodes (clades) on a phylogeny can be set using node.mins. This argument takes a vector of the same length as the number of nodes, with dates given in the same order as nodes are ordered in the `tree$edge` matrix. Note that in `tree$edge`, terminal tips are given the first set of numbers (`1:Ntip(tree)`), so the first element of `node.mins` is the first internal node (the node numbered `Ntip(tree)+1`, which is generally the root for most `phylo` objects read by `read.tree`). Not all nodes need be given minimum dates; those without minimum dates can be given as NA in `node.mins`, but the vector must be the same length as the number of internal nodes in `tree`. These are minimum date constraints, such that a node will be forced to be *at least as old as this date*, but the final date may be even older depending on the taxon dates used, the time-scaling method applied, the `vartime` used and any other minimum node dates given (e.g. if a clade is given a very old minimum date, this will (of course) over-ride any minimum dates given for clades that that node is nested within). Although `vartime` does adjust the node age downwards when the equal method is used, if a user has a specific date they'd like to constrain the root to, they should use `node.mins` instead because the result is more predictable.
`noisyDrop`	If `TRUE` (the default), any taxa dropped from tree due to not having a matching entry in the time data will be listed in a system message.
`plot`	If `TRUE`, plots the input and output phylogenies.
`timeList`	A list composed of two matrices giving interval times and taxon appearance dates. The rownames of the second matrix should be the taxon IDs, identical to the `tip.labels` for tree. See details.
`nonstoch.bin`	If `TRUE`, dates are not stochastically pulled from uniform distributions. See below for more details.
`sites`	Optional two column matrix, composed of site IDs for taxon FADs and LADs. The sites argument allows users to constrain the placement of dates by restricting multiple fossil taxa whose FADs or LADs are from the same very temporally restricted sites (such as fossil-rich Lagerstatten) to always have the same date, across many iterations of time-scaled trees. To do this, simply give a matrix where the "site" of each FAD and LAD for every taxon is listed, as corresponding to the second matrix in timeList. If no sites matrix is given (the default), then it is assumed all fossil come from different "sites" and there is no shared temporal structure among the events.
`point.occur`	If true, will automatically produce a 'sites' matrix which forces all FADs and LADs to equal each other. This should be used when all taxa are only known from single 'point occurrences', i.e. each is only recovered from a single bed/horizon, such as a Lagerstatten.

Details

Time-Scaling Methods

These functions are an attempt to unify and collect previously used and discussed 'a posteriori' methods for time-scaling phylogenies of fossil taxa. Unfortunately, it can be difficult to attribute some time-scaling methods to specific references in the literature.

There are five main a posteriori approaches that can be used by `timePaleoPhy`. Four of these main types use some value of absolute time, chosen a priori, to time-scale the tree. This is handled by the argument `vartime`, which is NULL by default and unused for type "basic".

"basic" This most simple of time-scaling methods ignores `vartime` and scales nodes so they are as old as the first appearance of their oldest descendant (Smith, 1994). This method produces many zero-length branches (Hunt and Carrano, 2010).

"equal" The 'equal' method defined by G. Lloyd and used in Brusatte et al. (2008) and Lloyd et al. (2012). Originally usable in code supplied by G. Lloyd, the algorithm is recreated here as closely as possible. This method works by increasing the time of the root divergence by some amount and then adjusting zero-length branches so that time on early branches is re-apportioned out along those later branches equally. Branches are adjusted in order relative to the number of nodes separating the edge from the root, going from the furthest (most shallow) edges to the deepest edges. The choice of ordering algorithm can have an unanticipated large effect on the resulting time-scaled trees created using "equal" and it appears that paleotree and functions written by G. Lloyd were not always consistent. The default option described here was only introduced into either software sources in August 2014. Thus, two legacy 'equal' methods are included in this function, so users can emulate older ordering algorithms for 'equal' which are now deprecated, as they do not match the underlying logic of the original 'equal' algorithm and do not minimize down-passes when adjusting branch lengths on the time-scaled tree.

The root age can be adjusted backwards in time by either increasing by an arbitrary amount (via the `vartime` argument) or by setting the root age directly (via the `node.mins` argument); conversely, the function will also allow a user to opt to not alter the root age at all.

"equal_paleotree_legacy" Exactly like 'equal' above, except that edges are ordered instead by their depth (i.e. number of nodes from the root). This minor modified version was referred to as 'equal' for this `timePaleoPhy` function in `paleotree` until February 2014, and thus is included here solely for legacy purposes. This ordering algorithm does not minimize branch adjustment cycles, like the newer default offered under currently 'equal'.

"equal_date.phylo_legacy" Exactly like 'equal' above, except that edges are ordered relative to their time (ie. total edge length) from the root following the application of the 'basic' time-scaling method, exactly as in G. Lloyd's original application. This was the method for sorting edges in the "equal" algorithm in G. Lloyd's `date.phylo` script and `DatePhylo` in package `strap` until August 2014, and was the default "equal" algorithm in `paleotree`'s `timePaleoPhy` function from February 2014 until August 2014. This ordering algorithm does not minimize branch adjustment cycles, like the newer default offered under currently 'equal'. Due to how the presence of zero-length branches can make ordering branches based on time to be very unpredictable, this version of the 'equal' algorithm is **highly not recommended**.

"aba" All branches additive. This method takes the "basic" tree and adds vartime to all branches. Note that this time-scaling method can warp the tree structure, leading to tips to originate out of order with the appearance data used.

"zlba" Zero-length branches additive. This method adds vartime to all zero-length branches in the "basic" tree. Discussed (possibly?) by Hunt and Carrano, 2010.Note that this time-scaling method can warp the tree structure, leading to tips to originate out of order with the appearance data used.

"mbl" Minimum branch length. Scales all branches so they are greater than or equal to vartime, and subtract time added to later branches from earlier branches in order to maintain the temporal structure of events. A version of this was first introduced by Laurin (2004).

These functions cannot time-scale branches relative to reconstructed character changes along branches, as used by Lloyd et al. (2012). Please see `DatePhylo` in R package `strap` for this functionality.

These functions will intuitively drop taxa from the tree with NA for range or are missing from `timeData` or `timeList`. Taxa dropped from the tree will be will be listed in a message output to the user. The same is done for taxa in the `timeList` object not listed in the tree.

As with many functions in the `paleotree` library, absolute time is always decreasing, i.e. the present day is zero.

As of August 2014, please note that the branch-ordering algorithm used in 'equal' has changed to match the current algorithm used by `DatePhylo` in package `strap`, and that two legacy versions of 'equal' have been added to this function, respectively representing how `timePaleoPhy` and `DatePhylo` (and its predecessor `date.phylo`) applied the 'equal' time-scaling method.

Interpretation of Taxon Ages in timePaleoPhy

`timePaleoPhy` is *primarily* designed for direct application to datasets where taxon first and last appearances are precisely known in continuous time, with no stratigraphic uncertainty. This is an uncommon form of data to have from the fossil record, although not an impossible form (micropaleontologists often have very precise range charts, for example). Instead, most data has some form of stratigraphic uncertainty. However, for some groups, the more typical 'first' and 'last' dates found in the literature or in databases represent the minimum and maximum absolute ages for the fossil collections that a taxon is known is known from. Presumably, the first and last appearances of that taxon in the fossil record is at unknown dates within these bounds.

As of paleotree version 2.0. the treatment of taxon ages in `timePaleoPhy` is handled by the argument `dateTreatment`. *By default,* this argument is set to 'firstLast' which means the matrix of ages are treated as precise first and last appearance dates (i.e. FADs and LADs). The earlier FADs will be used to calibrate the node ages, which could produce fairly nonsensical results if these are 'minimum' ages instead and reflect age uncertainty. Alternatively, `dateTreatment` can be set to 'minMax' which instead treats taxon age data as minimum and maximum bounds on a single point date. These point dates, if the minimum and maximum bounds option is selected, are chose under a uniform distribution. Many time-scaled trees should be created to approximate the uncertainty in the dates. Additionally, there is a third option for `dateTreatment`: users may also make it so that the 'times of observation' of trees are uncertain, such that the tips of the tree (with terminal ranges added) should be randomly selected from a uniform distribution. Essentially, this third option treats the dates as first and last appearances, but treats the first appearance dates as known and fixed, but the 'last appearance' dates as unknown. In previous versions of paleotree, this third option was enacted with the argument `rand.obs`, which has been removed for clarity.

Interpretation of Taxon Ages in bin_timePaleoPhy

As an alternative to using `timePaleoPhy`, `bin_timePaleoPhy` is a wrapper of `timePaleoPhy` which produces time-scaled trees for datasets which only have interval data available. For each output tree, taxon first and last appearance dates are placed within their listed intervals under a uniform distribution. Thus, a large sample of time-scaled trees will approximate the uncertainty in the actual timing of the FADs and LADs. In some ways, treating taxonomic age uncertainty may be more logical via `bin_timePaleoPhy`, as it is tied to specific interval bounds, and there are more options available for certain types of age uncertainty, such as for cases where specimens come from the same fossil site.

The input `timeList` object for `bin_timePaleoPhy` can have overlapping (i.e. non-sequential) intervals, and intervals of uneven size. Taxa alive in the modern should be listed as last occurring in a time interval that begins at time 0 and ends at time 0. If taxa occur only in single collections (i.e. their first and last appearance in the fossil record is synchronous, the argument point.occur will force all taxa to have instantaneous durations in the fossil record. Otherwise, by default, taxa are assumed to first and last appear in the fossil record at different points in time, with some positive duration. The sites matrix can be used to force only a portion of taxa to have simultaneous first and last appearances.

By setting the argument nonstoch.bin to TRUE for `bin_timePaleoPhy`, the dates are NOT stochastically pulled from uniform bins but instead FADs are assigned to the earliest time of whichever interval they were placed in and LADs are placed at the most recent time in their placed interval. This option may be useful for plotting. The sites argument becomes arbitrary if `nonstoch.bin = TRUE`.

If `timeData` or the elements of `timeList` are actually `data.frames` (as output by `read.csv` or `read.table`), these will be coerced to a matrix.

Tutorial

A tutorial for applying the time-scaling functions in paleotree, along with an example using real (graptolite) data, can be found here: http://nemagraptus.blogspot.com/2013/06/a-tutorial-to-cal3-time-scaling-using.html

Value

The output of these functions is a time-scaled tree or set of time-scaled trees, of either class `phylo` or `multiphylo`, depending on the argument ntrees. All trees are output with an element $root.time. This is the time of the root on the tree and is important for comparing patterns across trees. Note that the $root.time element is defined relative to the earliest first appearance date, and thus later tips may seem to occur in the distant future under the 'aba' and 'zbla' time-scaling methods.

Trees created with `bin_timePaleoPhy` will output with some additional elements, in particular $ranges.used, a matrix which records the continuous-time ranges generated for time-scaling each tree. (Essentially a pseudo-`timeData` matrix.)

Note

Please account for stratigraphic uncertainty in your analysis. Unless you have exceptionally resolved data, select an appropriate option in `dateTreatment` within `timePaleoPhy`, use the more sophisticated `bin_timePaleoPhy` or code your own wrapper function of `timePaleoPhy` that accounts for stratigraphic uncertainty in your dataset.

Author(s)

David W. Bapst, heavily inspired by code supplied by Graeme Lloyd and Gene Hunt.

References

Bapst, D. W. 2013. A stochastic rate-calibrated method for time-scaling phylogenies of fossil taxa. *Methods in Ecology and Evolution*. 4(8):724-733.

Bapst, D. W. 2014. Assessing the effect of time-scaling methods on phylogeny-based analyses in the fossil record. **Paleobiology 40**(3):331-351.

Brusatte, S. L., M. J. Benton, M. Ruta, and G. T. Lloyd. 2008 Superiority, Competition, and Opportunism in the Evolutionary Radiation of Dinosaurs. *Science* **321**(5895):1485-91488.

Hunt, G., and M. T. Carrano. 2010 Models and methods for analyzing phenotypic evolution in lineages and clades. In J. Alroy, and G. Hunt, eds. Short Course on Quantitative Methods in Paleobiology. Paleontological Society.

Laurin, M. 2004. The Evolution of Body Size, Cope's Rule and the Origin of Amniotes. *Systematic Biology* 53(4):594-622.

Lloyd, G. T., S. C. Wang, and S. L. Brusatte. 2012 Identifying Heterogeneity in Rates of Morphological Evolutio: Discrete Character Change in the Evolution of Lungfish(Sarcopterygii, Dipnoi). *Evolution* **66**(2):330–348.

Smith, A. B. 1994 Systematics and the fossil record: documenting evolutionary patterns. Blackwell Scientific, Oxford.

See Also

`cal3TimePaleoPhy`, `binTimeData`, `multi2di`

For an alternative time-scaling function, which includes the 'ruta' method that weights the time-scaling of branches by estimates of character change along with implementations of the 'basic' and 'equal' methods described here, please see function `DatePhylo` in package `strap`.

Examples

```
# examples with empirical data

#load data
data(retiolitinae)

#Can plot the unscaled cladogram
plot(retioTree)
#Can plot discrete time interval diversity curve with retioRanges
taxicDivDisc(retioRanges)

#Use basic time-scaling (terminal branches only go to FADs)
ttree<-bin_timePaleoPhy(tree=retioTree,timeList=retioRanges,type="basic",
ntrees=1, plot=TRUE)

#Use basic time-scaling (terminal branches go to LADs)
ttree<-bin_timePaleoPhy(tree=retioTree,timeList=retioRanges,type="basic",
add.term=TRUE, ntrees=1, plot=TRUE)

#mininum branch length time-scaling (terminal branches only go to FADs)
ttree<-bin_timePaleoPhy(tree=retioTree,timeList=retioRanges,type="mbl",
vartime=1, ntrees=1, plot=TRUE)

###################
```

```
# examples with simulated data

#Simulate some fossil ranges with simFossilRecord
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
#simulate a fossil record with imperfect sampling with sampleRanges
rangesCont <- sampleRanges(taxa,r=0.5)
#let's use taxa2cladogram to get the 'ideal' cladogram of the taxa
cladogram <- taxa2cladogram(taxa,plot=TRUE)
#Now let's try timePaleoPhy using the continuous range data
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",plot=TRUE)
#plot diversity curve
phyloDiv(ttree)

#that tree lacked the terminal parts of ranges (tips stops at the taxon FADs)
#let's add those terminal ranges back on with add.term
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",add.term=TRUE,plot=TRUE)
#plot diversity curve
phyloDiv(ttree)

#that tree didn't look very resolved, does it? (See Wagner and Erwin 1995 to see why)
#can randomly resolve trees using the argument randres
#each resulting tree will have polytomies randomly resolved in different ways using multi2di
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",ntrees=1,randres=TRUE,
    add.term=TRUE,plot=TRUE)
#notice well the warning it prints!
#we would need to set ntrees to a large number to get a fair sample of trees

#if we set ntrees>1, timePaleoPhy will make multiple time-trees
ttrees <- timePaleoPhy(cladogram,rangesCont,type="basic",ntrees=9,randres=TRUE,
    add.term=TRUE,plot=TRUE)
#let's compare nine of them at once in a plot
layout(matrix(1:9,3,3))
parOrig <- par(no.readonly=TRUE)
par(mar=c(1,1,1,1))
for(i in 1:9){plot(ladderize(ttrees[[i]]),show.tip.label=FALSE,no.margin=TRUE)}
#they are all a bit different!

#we can also resolve the polytomies in the tree according to time of first appearance
#via the function timeLadderTree, by setting the argument 'timeres' to TRUE
ttree <- timePaleoPhy(cladogram,rangesCont,type="basic",ntrees=1,timeres=TRUE,
    add.term=TRUE,plot=TRUE)

#can plot the median diversity curve with multiDiv
layout(1);par(parOrig)
multiDiv(ttrees)

#compare different methods of timePaleoPhy
layout(matrix(1:6,3,2))
parOrig <- par(no.readonly=TRUE)
par(mar=c(3,2,1,2))
```

```
plot(ladderize(timePaleoPhy(cladogram,rangesCont,type="basic",vartime=NULL,add.term=TRUE)))
    axisPhylo();text(x=50,y=23,"type=basic",adj=c(0,0.5),cex=1.2)
plot(ladderize(timePaleoPhy(cladogram,rangesCont,type="equal",vartime=10,add.term=TRUE)))
    axisPhylo();text(x=55,y=23,"type=equal",adj=c(0,0.5),cex=1.2)
plot(ladderize(timePaleoPhy(cladogram,rangesCont,type="aba",vartime=1,add.term=TRUE)))
    axisPhylo();text(x=55,y=23,"type=aba",adj=c(0,0.5),cex=1.2)
plot(ladderize(timePaleoPhy(cladogram,rangesCont,type="zlba",vartime=1,add.term=TRUE)))
    axisPhylo();text(x=55,y=23,"type=zlba",adj=c(0,0.5),cex=1.2)
plot(ladderize(timePaleoPhy(cladogram,rangesCont,type="mbl",vartime=1,add.term=TRUE)))
    axisPhylo();text(x=55,y=23,"type=mbl",adj=c(0,0.5),cex=1.2)
layout(1);par(parOrig)

#using node.mins
#let's say we have (molecular??) evidence that node #5 is at least 1200 time-units ago
#to use node.mins, first need to drop any unshared taxa
droppers <- cladogram$tip.label[is.na(
      match(cladogram$tip.label,names(which(!is.na(rangesCont[,1])))))]
cladoDrop <- drop.tip(cladogram, droppers)
# now make vector same length as number of nodes
nodeDates <- rep(NA, Nnode(cladoDrop))
nodeDates[5] <- 1200
ttree1 <- timePaleoPhy(cladoDrop,rangesCont,type="basic",
   randres=FALSE,node.mins=nodeDates,plot=TRUE)
ttree2 <- timePaleoPhy(cladoDrop,rangesCont,type="basic",
    randres=TRUE,node.mins=nodeDates,plot=TRUE)

#Using bin_timePaleoPhy to time-scale with discrete interval data
#first let's use binTimeData() to bin in intervals of 1 time unit
rangesDisc <- binTimeData(rangesCont,int.length=1)
ttreeB1 <- bin_timePaleoPhy(cladogram,rangesDisc,type="basic",ntrees=1,randres=TRUE,
    add.term=TRUE,plot=FALSE)
#notice the warning it prints!
phyloDiv(ttreeB1)
#with time-order resolving via timeLadderTree
ttreeB2 <- bin_timePaleoPhy(cladogram,rangesDisc,type="basic",ntrees=1,timeres=TRUE,
    add.term=TRUE,plot=FALSE)
phyloDiv(ttreeB2)
#can also force the appearance timings not to be chosen stochastically
ttreeB3 <- bin_timePaleoPhy(cladogram,rangesDisc,type="basic",ntrees=1,
    nonstoch.bin=TRUE,randres=TRUE,add.term=TRUE,plot=FALSE)
phyloDiv(ttreeB3)

# testing node.mins in bin_timePaleoPhy
ttree <- bin_timePaleoPhy(cladoDrop,rangesDisc,type="basic",ntrees=1,
    add.term=TRUE,randres=FALSE,node.mins=nodeDates,plot=TRUE)
# with randres = TRUE
ttree <- bin_timePaleoPhy(cladoDrop,rangesDisc,type="basic",ntrees=1,
    add.term=TRUE,randres=TRUE,node.mins=nodeDates,plot=TRUE)


#simple three taxon example for testing inc.term.adj
ranges1<-cbind(c(3,4,5),c(2,3,1));rownames(ranges1)<-paste("t",1:3,sep="")
clado1<-read.tree(file=NA,text="(t1,(t2,t3));")
```

```
ttree1<-timePaleoPhy(clado1,ranges1,type="mbl",vartime=1)
ttree2<-timePaleoPhy(clado1,ranges1,type="mbl",vartime=1,add.term=TRUE)
ttree3<-timePaleoPhy(clado1,ranges1,type="mbl",vartime=1,add.term=TRUE,inc.term.adj=TRUE)
layout(1:3)
ttree1$root.time;plot(ttree1);axisPhylo()
ttree2$root.time;plot(ttree2);axisPhylo()
ttree3$root.time;plot(ttree3);axisPhylo()
-apply(ranges1,1,diff)
```

timeSliceTree *Time-Slicing a Phylogeny*

Description

Removes the portion of a tree after a set point in time, as if the tree after that moment had been sliced away.

Usage

```
timeSliceTree(ttree, sliceTime, drop.extinct = FALSE, plot = TRUE)
```

Arguments

`ttree`	A time-scaled phylogeny of class phylo
`sliceTime`	Time to 'slice' the tree at. See details below.
`drop.extinct`	If true, drops tips that go extinct before timeSlice.
`plot`	If true, plots input and output trees for comparison.

Details

The function assumes that ttree will generally have an element called $root.time, which is the time before present that the root divergence occurred. If $root.time is not present as an element of ttree, then it is assumed the tip furthest from the root is at time 0 (present-day) and a new $root.time is calculated (a warning will be issued in this case).

The sliceTime is always calculated as on the same scale as ttree$root.time. In other words, if root.time=100, then timeSlice=80 will slice the tree 20 time units after the root.

If drop.extinct=T, then extinct tips are dropped and (if present) the $root.time of ttree is adjusted. This is done using the function dropExtinct.

Value

Returns the modified phylogeny as an object of class phylo.

Tip labels for cut branches which held multiple descendant tips will be the label for the earliest appearing tip descendant of that branch. This is somewhat arbitrary; the actual morphotaxon present at that time might have been a different taxon. For simulated datasets, use taxa2phylo.

Author(s)

David W. Bapst, with modification of code by Klaus Schliep to avoid use of function `dist.nodes` which has difficulty with large trees and greatly benefiting function runtime.

See Also

`phyloDiv`, `dropExtinct`, `dropExtant`

Also see the function treeSlice in the library phytools, which will slice a tree at some point in and return all the subtrees which remain after the slicing time. (Effectively the reverse of timeSliceTree.)

Examples

```
#a neat example of using phyloDiv with timeSliceTree
   #to simulate doing extant-only phylogeny studies
   #of diversification...in the past!
set.seed(444)
record<-simFossilRecord(p=0.1, q=0.1, nruns=1,
nTotalTaxa=c(30,40), nExtant=0)
taxa<-fossilRecord2fossilTaxa(record)
taxicDivCont(taxa)
#that's the whole diversity curve

#with timeSliceTree we could look at the lineage accumulation curve
   #we'd get of species sampled at a point in time
tree <- taxa2phylo(taxa)
#use timeSliceTree to make tree of relationships up until time=950
tree950 <- timeSliceTree(tree,sliceTime=950,plot=TRUE,drop.extinct=FALSE)

#use drop.extinct=T to only get the tree of lineages extant at time=950
tree950 <- timeSliceTree(tree,sliceTime=950,plot=FALSE,drop.extinct=TRUE)
#now its an ultrametric tree with many fewer tips...
#lets plot the lineage accumulation plot on a log scale
phyloDiv(tree950,plotLogRich=TRUE)
```

`unitLengthTree` *Scale Tree to Unit-Length*

Description

Rescales all edges of a phylogeny to be equal to 1 ("unit-length").

Usage

```
unitLengthTree(tree)
```

Arguments

`tree`	an object of class phylo

Details

Probably not a good way to scale a tree for comparative studies. What does it mean to scale every edge of the phylogeny to the same length?

Value

Returns the modified phylogeny as an object of class phylo. Any $root.time element is removed.

See Also

As an alternative to using unitLengthTree in comparative studies, see `timePaleoPhy`

See also `speciationalTree` in the package geiger, which does essentially the same thing as unitLengthTree.

Examples

```
set.seed(444)
tree <- rtree(10)
layout(1:2)
plot(tree)
plot(unitLengthTree(tree))
layout(1)
```

Index

∗Topic **datasets**
graptDisparity, 52
graptPBDB, 56
kanto, 64
macroperforateForam, 67
retiolitinae, 121
SongZhangDicrano, 150

ace, *112*
addTermBranchLength (modifyTerminalBranches), 90
ancPropStateMat, *119*
ancPropStateMat (minCharChange), 80
ape, *4*
as.function, *43*

bin_cal3TimePaleoPhy, *101*
bin_cal3TimePaleoPhy (cal3TimePaleoPhy), 8
bin_timePaleoPhy, *37*, *101*, *107*, *169*
bin_timePaleoPhy (timePaleoPhy), 170
bind.tip, *92*
bind.tree, *45*
bindPaleoTip (modifyTerminalBranches), 90
binTimeData, 5, *15*, *37*, *38*, *107*, *128*, *133*, *169*, *176*
branchClasses, 7

cal3 (cal3TimePaleoPhy), 8
cal3TimePaleoPhy, 8, *170*, *176*
candleTaxa (termTaxa), 161
charMatDicrano (SongZhangDicrano), 150
cladogeneticTraitCont, 17
cladogramDicrano (SongZhangDicrano), 150
cleanNewPhylo (testEdgeMat), 166
cleanTree (testEdgeMat), 166
collapseNodes (degradeTree), 31
communityEcology, 19
compareNodeAges (compareTimescaling), 22
compareTermBranches, *92*
compareTermBranches (compareTimescaling), 22
compareTimescaling, 22, *31*
constrainParPaleo, 24, *41*, *61*, *89*, *103*
contrasts, *82*, *119*
createMrBayesConstraints, 29

dateNodes, *23*, 30
deadTree (termTaxa), 161
degradeTree, 31
depthRainbow, 33
di2multi, *33*, *90*, *168*
divCurveFossilRecordSim, 34
DiversityCurves, 35, *95*, *107*
drop.tip, *92*
dropExtant, *180*
dropExtant (modifyTerminalBranches), 90
dropExtinct, *180*
dropExtinct (modifyTerminalBranches), 90
dropPaleoTip (modifyTerminalBranches), 90
dropZLB, *13*
dropZLB (modifyTerminalBranches), 90
durationFreq, 39, *59*, *63*, *133*

equation2function, 43
expandTaxonTree, 44

fixRootTime (modifyTerminalBranches), 90
footeValues, 46, *63*
foramAL (macroperforateForam), 67
foramALb (macroperforateForam), 67
foramAM (macroperforateForam), 67
foramAMb (macroperforateForam), 67
fossilRecord2fossilRanges (simFossilRecordMethods), 147
fossilRecord2fossilTaxa, *154*, *156*, *164*
fossilRecord2fossilTaxa (simFossilRecordMethods), 147

fourDate2timeList *(*timeList2fourDate*)*, 168
fourDateFunctions *(*timeList2fourDate*)*, 168
freqRat, *42*, 48, *63*, *133*

graptCharMatrix *(*graptDisparity*)*, 52
graptDisparity, 52, *99*, *122*
graptDistMat *(*graptDisparity*)*, 52
graptOccPBDB *(*graptPBDB*)*, 56
graptPBDB, 56, *76*, *101*, *111*, *158*
graptRanges *(*graptDisparity*)*, 52
graptTaxaPBDB *(*graptPBDB*)*, 56

horizonSampRate, 58
HurlbertPIE *(*communityEcology*)*, 19

inverseSurv, 59
invSurv *(*inverseSurv*)*, 59

kanto, *20*, 64

legend, *62*, *107*
ltt.plot, *38*

macroperforateForam, 67
make_durationFreqCont, *12*, *15*, *49*, *131*
make_durationFreqCont *(*durationFreq*)*, 39
make_durationFreqDisc, *12*, *49*, *131*
make_durationFreqDisc *(*durationFreq*)*, 39
make_inverseSurv *(*inverseSurv*)*, 59
makePBDBtaxonTree, *57*, 74, *105*, *161*
minBranchLength, 78
minCharChange, 80, *117*, *119*
modelMethods, *27*, *41*, *61*, 86, *103*
modifyTerminalBranches, 90
multi2di, *15*, *45*, *119*, *171*, *176*
multiDiv, *38*, 95

nearestNeighborDist, *53*, 98

occData2timeList, *57*, *76*, 100, *111*, *158*
optim, *27*, *88*, *103*
optimPaleo, 103

pairwiseSpearmanRho *(*communityEcology*)*, 19
paleotree *(*paleotree-package*)*, 3
paleotree-package, 3
parbounds *(*modelMethods*)*, 86
parbounds<- *(*modelMethods*)*, 86
parentChild2taxonTree, *76*, 104, *161*
parInit, *27*, *103*
parInit *(*modelMethods*)*, 86
parLower, *103*
parLower *(*modelMethods*)*, 86
parLower<- *(*modelMethods*)*, 86
parnames, *25*, *27*
parnames *(*modelMethods*)*, 86
parnames<- *(*modelMethods*)*, 86
parUpper, *103*
parUpper *(*modelMethods*)*, 86
parUpper<- *(*modelMethods*)*, 86
perCapitaRates, 106
perfectParsCharTree, 108
phyloDiv, *23*, *92*, *97*, *180*
phyloDiv *(*DiversityCurves*)*, 35
plotMultiDiv *(*multiDiv*)*, 95
plotOccData, *57*, *101*, 110, *158*
plotTraitgram, 111
pqr2Ps, 113, *131*
pqsRate2sProb *(*SamplingConv*)*, 129
probAnc, 115, *131*

qsProb2Comp, *42*, *49*, *63*
qsProb2Comp *(*SamplingConv*)*, 129
qsRate2Comp, *42*, *63*
qsRate2Comp *(*SamplingConv*)*, 129

resolveTreeChar, 116
retioChar *(*retiolitinae*)*, 121
retiolitinae, *53*, 121
retioRanges *(*retiolitinae*)*, 121
retioTree *(*retiolitinae*)*, 121
reverseList, 123
rootSplit, 124

sampleRanges, *6*, 125, *131*
SamplingConv, *41*, *59*, *63*, *107*, *114*, *116*, 129
seqTimeList, 131
simCandleTaxa *(*termTaxa*)*, 161
simFossilRecord, *6*, *18*, *35*, *128*, 133, *149*, *154–156*, *164*
simFossilRecordMethods, *140*, 147
simTermTaxa *(*termTaxa*)*, 161
simTermTaxaAdvanced *(*termTaxa*)*, 161
SongZhangDicrano, 150
sProb2sRate, *12*, *42*, *49*, *63*
sProb2sRate *(*SamplingConv*)*, 129

sRate2sProb, *42*, *63*
sRate2sProb *(*SamplingConv*)*, 129

taxa2cladogram, 152, *156*, *164*
taxa2phylo, *23*, *154*, 154, *164*
taxicDivCont, *6*, *97*
taxicDivCont *(*DiversityCurves*)*, 35
taxicDivDisc, *97*, *101*, *169*
taxicDivDisc *(*DiversityCurves*)*, 35
taxonSortPBDBocc, *57*, *76*, *100*, *101*, *110*, *111*, 156
taxonTable2taxonTree, *76*, *105*, 160
termTaxa, 161
testEdgeMat, 165
timeLadderTree, *33*, 167, *171*
timeList2fourDate, 168
timePaleoPhy, *15*, *79*, 170, *181*
timeSliceFossilRecord *(*simFossilRecordMethods*)*, 147
timeSliceTree, *38*, 179
trueCandle *(*termTaxa*)*, 161
trueTermTaxaTree *(*termTaxa*)*, 161

unitLengthTree, *18*, 180