Package ‘castor’

July 30, 2019

Type Package

Title Efficient Phylogenetics on Large Trees

Version 1.4.2

Date 2019-07-29

Author Stilianos Louca

Maintainer Stilianos Louca `<louca@zoology.ubc.ca>`

Description Efficient phylogenetic analyses on massive phylogenies comprising up to millions of tips. Functions include pruning, rerooting, calculation of most-recent common ancestors, calculating distances from the tree root and calculating pairwise distances. Calculation of phylogenetic signal and mean trait depth (trait conservatism), ancestral state reconstruction and hidden character prediction of discrete characters, simulating and fitting models of trait evolution, fitting and simulating diversification models, dating trees, comparing trees, and reading/writing trees in Newick format. Citation: Louca, Stilianos and Doebeli, Michael (2017) <doi:10.1093/bioinformatics/btx701>.

License GPL (>= 2)

Depends Rcpp (>= 0.12.10)

Imports parallel, naturalsort, stats, nloptr

LinkingTo Rcpp

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-07-29 22:40:02 UTC

R topics documented:

`castor-package` *Efficient computations on large phylogenetic trees.*

Description

This package provides efficient tree manipulation functions including pruning, rerooting, calculation of most-recent common ancestors, calculating distances from the tree root and calculating pairwise distance matrices. Calculation of phylogenetic signal and mean trait depth (trait conservatism). Efficient ancestral state reconstruction and hidden character prediction of discrete characters on phylogenetic trees, using Maximum Likelihood and Maximum Parsimony methods. Simulating models of trait evolution, and generating random trees.

Details

The most important data unit is a phylogenetic tree of class "phylo", with the tree topology encoded in the member variable `tree.edge`. See the `ape` package manual for details on the "phylo" format. The castor package was designed to be efficient for large phylogenetic trees (>10,000 tips), and scales well to trees with millions of tips. Most functions have asymptotically linear time complexity O(N) in the number of edges N. This efficiency is achived via temporary auxiliary data structures, use of dynamic programing, heavy use of C++, and integer-based indexing instead of name-based indexing of arrays. All functions support trees that include monofurcations (nodes with a single child) as well as multifurcations (nodes with more than 2 children). See the associated paper by Louca et al. for a comparison with other packages.

Throughout this manual, "Ntips" refers to the number of tips, "Nnodes" to the number of nodes and "Nedges" to the number of edges in a tree. In the context of discrete trait evolution/reconstruction, "Nstates" refers to the number of possible states of the trait. In the context of multivariate trait evolution, "Ntraits" refers to the number of traits.

Author(s)

Stilianos Louca

Maintainer: Stilianos Louca <louca@zoology.ubc.ca>

References

S. Louca and M. Doebeli (2017). Efficient comparative phylogenetics on large trees. Bioinformatics. DOI:10.1093/bioinformatics/btx701

`asr_empirical_probabilities` *Empirical ancestral state probabilities.*

Description

Given a rooted phylogenetic tree and the states of a discrete trait for each tip, calculate the empirical state frequencies/probabilities for each node in the tree, i.e. the frequencies/probabilities of states across all tips descending from that node. This may be used as a very crude estimate of ancestral state probabilities.

Usage

```
asr_empirical_probabilities(tree, tip_states, Nstates=NULL,
                            probabilities=TRUE, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree as an integer from 1 to Nstates, where Nstates is the possible number of states (see below).
`Nstates`	Either `NULL`, or an integer specifying the number of possible states of the trait. If `NULL`, then it will be computed based on the maximum value encountered in `tip_states`
`probabilities`	Logical, specifying whether empirical frequencies should be normalized to represent probabilities. If `FALSE`, then the raw occurrence counts are returned.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g., characters or factors), you should map them to a set of integers 1,..,Nstates. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). The function has asymptotic time complexity O(Nedges x Nstates).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

Value

A list with the following elements:

`ancestral_likelihoods`
: A 2D integer (if `probabilities==FALSE`) or numeric (if `probabilities==TRUE`) matrix, listing the frequency or probability of each state for each node. This matrix will have size Nnodes x Nstates, where Nstates was either explicitly provided as an argument or inferred from `tip_states`. The rows in this matrix will be in the order in which nodes are indexed in the tree, i.e. the [n,s]-th entry will be the frequency or probability of the s-th state for the n-th node. Note that the name was chosen for compatibility with other ASR functions.

Author(s)

Stilianos Louca

See Also

`asr_max_parsimony`, `asr_squared_change_parsimony` `asr_mk_model`, `map_to_state_space`

Examples

```
## Not run:
asr_empirical_probabilities(tree, tip_states=c(1,3,2,3), Nstates=3)

## End(Not run)
```

---

asr_independent_contrasts

Ancestral state reconstruction via phylogenetic independent contrasts.

Description

Reconstruct ancestral states for a continuous (numeric) trait using phylogenetic independent contrasts (PIC; Felsenstein, 1985).

Usage

```
asr_independent_contrasts(tree,
                          tip_states,
                          weighted    = TRUE,
                          include_CI  = FALSE,
                          check_input = TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the known state of each tip in the tree.
`weighted`	Logical, specifying whether to weight tips and nodes by the inverse length of their incoming edge, as in the original method by Felsenstein (1985). If `FALSE`, edge lengths are treated as if they were 1.
`include_CI`	Logical, specifying whether to also calculate standard errors and confidence intervals for the reconstructed states under a Brownian motion model, as described by Garland et al (1999).
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

The function traverses the tree in postorder (tips–>root) and estimates the state of each node as a convex combination of the estimated states of its chilren. These estimates are the intermediate "X" variables introduced by Felsenstein (1985) in his phylogenetic independent contrasts method. For the root, this yields the same globally parsimonious state as the squared-changes parsimony algorithm implemented in `asr_squared_change_parsimony` (Maddison 1991). For any other node, PIC only yields locally parsimonious reconstructions, i.e. reconstructed states only depend on the subtree descending from the node (see discussion by Maddison 1991).

If `weighted==TRUE`, then this function yields the same ancestral state reconstructions as

`ape::ace(phy=tree,x=tip_states,type="continuous",method="pic",model="BM",CI=FALSE)`

in the `ape` package (v. 0.5-64). Note that in contrast to the CI95 returned by `ape::ace`, the confidence intervals calculated here have the same units as the trait and depend both on the tree topology as well as the tip states.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. This is the same as setting `weighted=FALSE`. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). Edges with length 0 will be adjusted internally to some tiny length if needed (if `weighted==TRUE`).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`). All tip states must be non-NA; otherwise, consider using one of the functions for hidden-state-prediction (e.g., `hsp_independent_contrasts`).

The function has asymptotic time complexity O(Nedges).

Value

A list with the following elements:

`ancestral_states`
: A numeric vector of size Nnodes, listing the reconstructed state of each node. The entries in this vector will be in the order in which nodes are indexed in the tree.

`standard_errors`
: Numeric vector of size Nnodes, listing the phylogenetically estimated standard error for the state in each node, under a Brownian motion model. The standard errors have the same units as the trait and depend both on the tree topology as well as the tip states. Calculated as described by Garland et al. (1999, page 377). Only included if `include_CI==TRUE`.

`CI95`
: Numeric vector of size Nnodes, listing the radius (half width) of the 95% confidence interval of the state in each node. Confidence intervals have same units as the trait and depend both on the tree topology as well as the tip states. For each node, the confidence interval is calculated according to the Student's t-distribution with Npics degrees of freedom, where Npics is the number of internally calculated independent contrasts descending from the node [Garland et al, 1999]. Only included if `include_CI==TRUE`.

Author(s)

Stilianos Louca

References

J. Felsenstein (1985). Phylogenies and the Comparative Method. The American Naturalist. 125:1-15.

W. P. Maddison (1991). Squared-change parsimony reconstructions of ancestral states for continuous-valued characters on a phylogenetic tree. Systematic Zoology. 40:304-314.

T. Garland Jr., P. E. Midford, A. R. Ives (1999). An introduction to phylogenetically based statistical methods, with a new method for confidence intervals on ancestral values. American Zoologist. 39:374-388.

See Also

`asr_squared_change_parsimony`, `asr_max_parsimony`, `asr_mk_model`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# reconstruct node states via weighted PIC
```

```
asr = asr_independent_contrasts(tree, tip_states, weighted=TRUE, include_CI=TRUE)
node_states = asr$ancestral_states

# get lower bounds of 95% CIs
lower_bounds = node_states - asr$CI95
```

`asr_max_parsimony` *Maximum-parsimony ancestral state reconstruction.*

Description

Reconstruct ancestral states for a discrete trait using maximum parsimony. Transition costs can vary between transitions, and can optionally be weighted by edge length.

Usage

```
asr_max_parsimony(tree, tip_states, Nstates=NULL,
                    transition_costs="all_equal",
                    edge_exponent=0, weight_by_scenarios=TRUE,
                    check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree as an integer from 1 to Nstates, where Nstates is the possible number of states (see below).
`Nstates`	Either `NULL`, or an integer specifying the number of possible states of the trait. If `NULL`, then Nstates will be computed based on the maximum value encountered in `tip_states`
`transition_costs`	Either "all_equal","sequential", "proportional", "exponential", or a quadratic non-negatively valued matrix of size Nstates x Nstates, specifying the transition costs between all possible states (which can include 0 as well as `Inf`). The [r,c]-th entry of the matrix is the cost of transitioning from state r to state c. The option "all_equal" specifies that all transitions are permitted and are equally costly. "sequential" means that only transitions between adjacent states are permitted and are all equally costly. "proportional" means that all transitions are permitted, but the cost increases proportional to the distance between states. "exponential" means that all transitions are permitted, but the cost increases exponentially with the distance between states. The options "sequential" and "proportional" only make sense if states exhibit an order relation (as reflected in their integer representation).

`edge_exponent`
: Non-negative real-valued number. Optional exponent for weighting transition costs by the inverse length of edge lengths. If 0, edge lengths do not influence the ancestral state reconstruction (this is the conventional max-parsimony). If >0, then at each edge the transition costs are multiplied by $1/L^e$, where L is the edge length and e is the edge exponent. This parameter is mostly experimental; modify at your own discretion.

`weight_by_scenarios`
: Logical, indicating whether to weight each optimal state of a node by the number of optimal maximum-parsimony scenarios in which the node is in that state. If FALSE, then all optimal states of a node are weighted equally (i.e. are assigned equal probabilities).

`check_input`
: Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. The order of states (if relevant) should be reflected in their integer representation. For example, if your original states are "small", "medium" and "large" and `transition_costs=="sequential"`, it is advised to represent these states as integers 1,2,3. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

This function utilizes Sankoff's (1975) dynamic programming algorithm for determining the smallest number (or least costly if transition costs are uneven) of state changes along edges needed to reproduce the observed tip states. The function has asymptotic time complexity O(Ntips+Nnodes x Nstates).

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. If `edge_exponent` is 0, then edge lengths do not influence the result. If `edge_exponent!=0`, then all edges must have non-zero length. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. None of the input vectors or matrixes need include row or column names; if they do, however, they are checked for consistency (if `check_input==TRUE`).

This function is meant for reconstructing ancestral states in all nodes of a tree, when the state of each tip is known. If some of the tips have unknown state, consider either pruning the tree to keep only tips with known states, or using the function `hsp_max_parsimony`.

Value

A list with the following elements:

`success`
: Boolean, indicating whether ASR was successful. If `FALSE`, the remaining returned elements may be undefined.

`ancestral_likelihoods`
A 2D numeric matrix, listing the probability of each node being in each state. This matrix will have size Nnodes x Nstates, where Nstates was either explicitly provided as an argument or inferred from `tip_states`. The rows in this matrix will be in the order in which nodes are indexed in the tree, i.e. the [n,s]-th entry will be the probability of the s-th state for the n-th node. Note that the name was chosen for compatibility with other ASR functions.

`total_cost` Real number, specifying the total transition cost across the tree for the most parsimonious scenario. In the classical case where `transition_costs="all_equal"`, the `total_cost` equals the total number of state changes in the tree under the most parsimonious scenario.

Author(s)

Stilianos Louca

References

D. Sankoff (1975). Minimal mutation trees of sequences. SIAM Journal of Applied Mathematics. 28:35-42.

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

See Also

`hsp_max_parsimony`, `asr_squared_change_parsimony` `asr_mk_model`, `hsp_mk_model`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 10
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a discrete trait
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ER")
tip_states = simulate_mk_model(tree, Q)$tip_states

# reconstruct node states via MPR
results = asr_max_parsimony(tree, tip_states, Nstates)
node_states = max.col(results$ancestral_likelihoods)

# print reconstructed node states
print(node_states)
```

asr_mk_model *Ancestral state reconstruction with Mk models and rerooting*

Description

Ancestral state reconstruction of a discrete trait using a fixed-rates continuous-time Markov model (a.k.a. "Mk model"). This function can estimate the (instantaneous) transition matrix using maximum likelihood, or take a specified transition matrix. The function can optionally calculate marginal ancestral state likelihoods for each node in the tree, using the rerooting method by Yang et al. (1995).

Usage

```
asr_mk_model( tree,
              tip_states,
              Nstates = NULL,
              tip_priors = NULL,
              rate_model = "ER",
              transition_matrix = NULL,
              include_ancestral_likelihoods = TRUE,
              reroot = TRUE,
              root_prior = "empirical",
              Ntrials = 1,
              optim_algorithm = "nlminb",
              optim_max_iterations = 200,
              optim_rel_tol = 1e-8,
              store_exponentials = TRUE,
              check_input =TRUE,
              Nthreads = 1)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree in terms of an integer from 1 to Nstates, where Ntips is the number of tips and Nstates is the number of possible states (see below). Can also be `NULL`. If `tip_states==NULL`, then `tip_priors` must not be `NULL` (see below).
`Nstates`	Either NULL, or an integer specifying the number of possible states of the trait. If `Nstates==NULL`, then it will be computed based on the maximum value encountered in `tip_states` or based on the number of columns in `tip_priors` (whichever is non-`NULL`).
`tip_priors`	A 2D numeric matrix of size Ntips x Nstates, where Nstates is the possible number of states for the character modelled. Can also be `NULL`. Each row of this matrix must be a probability vector, i.e. it must only contain non-negative entries and must sum up to 1. The [i,s]-th entry should be the prior probability

of tip i being in state s. If you know for certain that tip i is in some state s, you can set the corresponding entry to 1 and all other entries in that row to 0. If you know the exact states of all tips, you can also pass these via `tip_states` instead. If `tip_priors==NULL`, then `tip_states` must not be `NULL` (see above).

`rate_model` Rate model to be used for fitting the transition rate matrix. Can be "ER" (all rates equal), "SYM" (transition rate i–>j is equal to transition rate j–>i), "ARD" (all rates can be different), "SUEDE" (only stepwise transitions i–>i+1 and i–>i-1 allowed, all 'up' transitions are equal, all 'down' transitions are equal) or "SRD" (only stepwise transitions i–>i+1 and i–>i-1 allowed, and each rate can be different). Can also be an index matrix that maps entries of the transition matrix to the corresponding independent rate parameter to be fitted. Diagonal entries should map to 0, since diagonal entries are not treated as independent rate parameters but are calculated from the remaining entries in the transition matrix. All other entries that map to 0 represent a transition rate of zero. The format of this index matrix is similar to the format used by the `ace` function in the `ape` package. `rate_model` is only relevant if `transition_matrix==NULL`.

`transition_matrix`
Either a numeric quadratic matrix of size Nstates x Nstates containing fixed transition rates, or `NULL`. The [r,c]-th entry in this matrix should store the transition rate from state r to state c. Each row in this matrix must have sum zero. If `NULL`, then the transition rates will be estimated using maximum likelihood, based on the `rate_model` specified.

`root_prior` Prior probability distribution of the root's states, used to calculate the model's overall likelihood from the root's marginal ancestral state likelihoods. Can be "flat" (all states equal), "empirical" (empirical probability distribution of states across the tree's tips) or "stationary" (stationary probability distribution of the transition matrix). If "stationary" and `transition_matrix==NULL`, then a transition matrix is first fitted using a flat root prior, and then used to calculate the stationary distribution. `root_prior` can also be a non-negative numeric vector of size Nstates and with total sum equal to 1.

`include_ancestral_likelihoods`
Include the marginal ancestral likelihoods for each node (conditional scaled state likelihoods) in the return values. Note that this may increase the computation time and memory needed, so you may set this to `FALSE` if you don't need marginal ancestral states.

`reroot` Reroot tree at each node when computing marginal ancestral likelihoods, according to Yang et al. (1995). This is the default and recommended behavior, but leads to increased computation time. If `FALSE`, ancestral likelihoods at each node are computed solely based on the subtree descending from that node, without rerooting.

`Ntrials` Number of trials (starting points) for fitting the transition matrix. Only relevant if `transition_matrix=NULL`. A higher number may reduce the risk of landing in a local non-global optimum of the likelihood function, but will increase computation time during fitting.

`optim_algorithm`
Either "optim" or "nlminb", specifying which optimization algorithm to use

for maximum-likelihood estimation of the transition matrix. Only relevant if `transition_matrix==NULL`.

`optim_max_iterations`
: Maximum number of iterations (per fitting trial) allowed for optimizing the likelihood function.

`optim_rel_tol`
: Relative tolerance (stop criterion) for optimizing the likelihood function.

`store_exponentials`
: Logical, specifying whether to pre-calculate and store exponentials of the transition matrix during calculation of ancestral likelihoods. This may reduce computation time because each exponential is only calculated once, but requires more memory since all exponentials are stored.

  Only relevant if `include_ancestral_likelihoods==TRUE`, otherwise exponentials are never stored.

`check_input`
: Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

`Nthreads`
: Number of parallel threads to use for running multiple fitting trials simultaneously. This only makes sense if your computer has multiple cores/CPUs and if `Ntrials>1`, and is only relevant if `transition_matrix==NULL`. This option is ignored on Windows, because Windows does not support forking.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. The order of states (if relevant) should be reflected in their integer representation. For example, if your original states are "small", "medium" and "large" and `rate_model=="SUEDE"`, it is advised to represent these states as integers 1,2,3. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

This function allows the specification of the precise tip states (if these are known) using the vector `tip_states`. Alternatively, if some tip states are only known in terms of a probability distribution, you can pass these probability distributions using the matrix `tip_priors`. Note that exactly one of the two arguments, `tip_states` or `tip_priors`, must be non-`NULL`.

Tips must be represented in `tip_states` or `tip_priors` in the same order as in `tree$tip.label`. None of the input vectors or matrixes need include row or column names; if they do, however, they are checked for consistency (if `check_input==TRUE`).

The tree is either assumed to be complete (i.e. include all possible species), or to represent a random subset of species chosen independently of their states. The rerooting method by Yang et al (1995) is used to calculate the marginal ancestral state likelihoods for each node by treating the node as a root and calculating its conditional scaled likelihoods. Note that the re-rooting algorithm is strictly speaking only valid for reversible Mk models, that is, satisfying the criterion

$$\pi_i Q_{ij} = \pi_j Q_{ji}, \quad \forall i, j,$$

where Q is the transition rate matrix and π is the stationary distribution of the model. The rate models "ER", 'SYM", "SUEDE" and "SRD" are reversible. For example, for "SUEDE" or "SRD" choose $\pi_{i+1} = \pi_i Q_{i,i+1}/Q_{i+1,i}$. In contrast, "ARD" models are generally not reversible.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). This function is similar to `rerootingMethod` in the `phytools` package (v0.5-64) and similar to `ape::ace` (v4.1) with options `method="ML",type="discrete"` and `marginal=FALSE`, but tends to be much faster than `rerootingMethod` and `ace` for large trees.

Value

A list with the following elements:

`success`	Logical, indicating whether ASR was successful. If `FALSE`, all other return values may be `NULL`.
`Nstates`	Integer, specifying the number of modeled trait states.
`transition_matrix`	A numeric quadratic matrix of size Nstates x Nstates, containing the transition rates of the Markov model. The [r,c]-th entry is the transition rate from state r to state c. Will be the same as the input `transition_matrix`, if the latter was not `NULL`.
`loglikelihood`	Log-likelihood of the Markov model for the observed tip states. If `transition_matrix` was `NULL` in the input, then this will be the log-likelihood maximized during fitting.
`ancestral_likelihoods`	Optional, only returned if `include_ancestral_likelihoods` was `TRUE`. A 2D numeric matrix, listing the likelihood of each state at each node (marginal ancestral likelihoods). This matrix will have size Nnodes x Nstates, where Nstates was either explicitly provided as an argument, or inferred from `tip_states` or `tip_priors` (whichever was non-`NULL`). The rows in this matrix will be in the order in which nodes are indexed in the tree, i.e. the [n,s]-th entry will be the likelihood of the s-th state at the n-th node. For example, `likelihoods[1,3]` will store the likelihood of observing the tree's tip states (if `reroot=TRUE`) or the descending subtree's tip states (if `reroot=FALSE`), if the first node was in state 3. Note that likelihoods are rescaled (normalized) to sum to 1 for convenience and numerical stability. The marginal likelihoods at a node should not, however, be interpreted as a probability distribution among states.

Author(s)

Stilianos Louca

References

Z. Yang, S. Kumar and M. Nei (1995). A new method for inference of ancestral nucleotide and amino acid sequences. Genetics. 141:1641-1650.

See Also

`hsp_mk_model`, `asr_max_parsimony`, `asr_squared_change_parsimony`, `hsp_max_parsimony`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 1000
tree  = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# create random transition matrix
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ER", max_rate=0.01)
cat(sprintf("Simulated ER transition rate=%g\n",Q[1,2]))

# simulate the trait's evolution
simulation = simulate_mk_model(tree, Q)
tip_states = simulation$tip_states
cat(sprintf("Simulated states for last 20 nodes:\n"))
print(tail(simulation$node_states,20))

# reconstruct node states from simulated tip states
# at each node, pick state with highest marginal likelihood
results = asr_mk_model(tree, tip_states, Nstates, rate_model="ER", Ntrials=2)
node_states = max.col(results$ancestral_likelihoods)

# print Mk model fitting summary
cat(sprintf("Mk model: log-likelihood=%g\n",results$loglikelihood))
cat(sprintf("Fitted ER transition rate=%g\n",results$transition_matrix[1,2]))

# print reconstructed node states for last 20 nodes
print(tail(node_states,20))
```

---

asr_squared_change_parsimony

Squared-change parsimony ancestral state reconstruction.

Description

Reconstruct ancestral states for a continuous (numeric) trait using squared-change maximum parsimony (Maddison, 1991). Transition costs can optionally be weighted by the inverse edge lengths ("weighted squared-change parsimony" by Maddison).

Usage

```
asr_squared_change_parsimony(tree, tip_states,  weighted=TRUE, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the known state of each tip in the tree.

`weighted`	Logical, specifying whether to weight transition costs by the inverted edge lengths. This corresponds to the "weighted squared-change parsimony" reconstruction by Maddison (1991) for a Brownian motion model of trait evolution.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

The function traverses the tree in postorder (tips–>root) to calculate the quadratic parameters described by Maddison (1991) and obtain the globally parsimonious squared-change parsimony state for the root. The function then reroots at each node, updates all affected quadratic parameters in the tree and calculates the node's globally parsimonious squared-change parsimony state. The function has asymptotic time complexity O(Nedges).

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. This is the same as setting `weighted=FALSE`. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). Edges with length 0 will be adjusted internally to some tiny length if needed (if `weighted==TRUE`).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

If `weighted==FALSE`, then this function yields the same ancestral state reconstructions as

`ape::ace(tip_states,tree,type="continuous",method="ML",model="BM",CI=FALSE)`

in the `ape` package (v. 0.5-64), assuming the tree as unit edge lengths. If `weighted==TRUE`, then this function yields the same ancestral state reconstructions as the maximum likelihood estimates under a Brownian motion model, as implemented by the `Rphylopars` package (v. 0.2.10):

`Rphylopars::anc.recon(tip_states,tree,vars=FALSE,CI=FALSE)`.

Value

A list with the following elements:

`ancestral_states`	A numeric vector of size Nnodes, listing the reconstructed state of each node. The entries in this vector will be in the order in which nodes are indexed in the tree.
`total_sum_of_squared_changes`	The total sum of squared changes, minimized by the (optionally weighted) squared-change parsimony algorithm. This is equation 7 in (Maddison, 1991).

Author(s)

Stilianos Louca

References

W. P. Maddison (1991). Squared-change parsimony reconstructions of ancestral states for continuous-valued characters on a phylogenetic tree. Systematic Zoology. 40:304-314.

See Also

`asr_independent_contrasts` `asr_max_parsimony`, `asr_mk_model`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# reconstruct node states based on simulated tip states
node_states = asr_squared_change_parsimony(tree, tip_states, weighted=TRUE)$ancestral_states
```

`asr_subtree_averaging` *Ancestral state reconstruction via subtree averaging.*

Description

Reconstruct ancestral states in a phylogenetic tree for a continuous (numeric) trait by averaging trait values over descending subtrees. That is, for each node the reconstructed state is set to the arithmetic average state of all tips descending from that node.

Usage

```
asr_subtree_averaging(tree, tip_states, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the known state of each tip in the tree.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

The function returns the estimated ancestral states (=averages) as well as the corresponding standard deviations. Note that reconstructed states are local estimates, i.e. they only take into account the tips descending from the reconstructed node.

The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). Edge lengths and distances between tips and nodes are not taken into account. All tip states are assumed to be known, and `NA` or `NaN` are not allowed in `tip_states`.

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

Value

A list with the following elements:

`success`	Logical, indicating whether ASR was sucessful. If all input data are valid then this will always be `TRUE`, but it is provided for consistency with other ASR functions.
`ancestral_states`	A numeric vector of size Nnodes, listing the reconstructed state (=average over descending tips) for each node. The entries in this vector will be in the order in which nodes are indexed in the tree.
`ancestral_stds`	A numeric vector of size Nnodes, listing the standard deviations corresponding to `ancestral_stds`.
`ancestral_counts`	A numeric vector of size Nnodes, listing the number of (descending) tips used to reconstruct the state of each node.

Author(s)

Stilianos Louca

See Also

`asr_independent_contrasts`, `asr_squared_change_parsimony`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# reconstruct node states by averaging simulated tip states
node_states = asr_subtree_averaging(tree, tip_states)$ancestral_states
```

`collapse_monofurcations` *Remove monofurcations from a tree.*

Description

Eliminate monofurcations (nodes with only a single child) from a phylogenetic tree, by connecting their incoming and outgoing edge.

Usage

```
collapse_monofurcations(tree, force_keep_root=TRUE, as_edge_counts=FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo".
`force_keep_root`	Logical, indicating whether the root node should always be kept (i.e., even if it only has a single child).
`as_edge_counts`	Logical, indicating whether all edges should be assumed to have length 1. If `TRUE`, the outcome is the same as if the tree had no edges.

Details

All tips in the input tree retain their original indices, however the returned tree may include fewer nodes and edges. Edge and node indices may change.

If `tree$edge.length` is missing, then all edges in the input tree are assumed to have length 1.

Value

A list with the following elements:

`tree`	A new tree of class "phylo", containing only bifurcations (and multifurcations, if these existed in the input tree). The number of nodes in this tree, Nnodes_new, may be lower than of the input tree.
`new2old_node`	Integer vector of length Nnodes_new, mapping node indices in the new tree to node indices in the old tree.
`Nnodes_removed`	Integer. Number of nodes (monofurcations) removed from the tree.

Author(s)

Stilianos Louca

See Also

`multifurcations_to_bifurcations`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1), max_tips=Ntips)$tree

# prune the tree to generate random monofurcations
random_tips = sample.int(n=Ntips, size=0.5 * Ntips, replace=FALSE)
tree = get_subtree_with_tips(tree, only_tips=random_tips, collapse_monofurcations=FALSE)$sub

# collapse monofurcations
new_tree = collapse_monofurcations(tree)$tree

# print summary of old and new tree
cat(sprintf("Old tree has %d nodes\n",tree$Nnode))
cat(sprintf("New tree has %d nodes\n",new_tree$Nnode))
```

---

collapse_tree_at_resolution

Collapse nodes of a tree at a phylogenetic resolution.

Description

Given a rooted tree and a phylogenetic resolution threshold, collapse all nodes whose distance to all descending tips does not exceed the threshold (or whose sum of descending edge lengths does not exceed the threshold), into new tips. This function can be used to obtain a "coarser" version of the tree, or to cluster closely related tips into a single tip.

Usage

```
collapse_tree_at_resolution(tree,
                            resolution              = 0,
                            by_edge_count           = FALSE,
                            shorten                 = TRUE,
                            rename_collapsed_nodes  = FALSE,
                            criterion               = 'max_tip_depth')
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`resolution`	Numeric, specifying the phylogenetic resolution at which to collapse the tree. This is the maximum distance a descending tip can have from a node, such that the node is collapsed into a new tip. If set to 0 (default), then only nodes whose descending tips are identical to the node will be collapsed.
`by_edge_count`	Logical. Instead of considering edge lengths, consider edge counts as phylogenetic distance between nodes and tips. This is the same as if all edges had length equal to 1.

`shorten`	Logical, indicating whether collapsed nodes should be turned into tips at the same location (thus potentially shortening the tree). If `FALSE`, then the incoming edge of each collapsed node is extended by some length L, where L is the distance of the node to its farthest descending tip (thus maintaining the height of the tree).
`rename_collapsed_nodes`	Logical, indicating whether collapsed nodes should be renamed using a representative tip name (the farthest descending tip).
`criterion`	Character, specifying the criterion to use for collapsing (i.e. how to interpret `resolution`). '`max_tip_depth`': Collapse nodes based on their maximum distance to any descending tip. '`sum_tip_paths`': Collapse nodes based on the sum of descending edges (each edge counted once). '`max_tip_pair_dist`': Collapse nodes based on the maximum distance between any pair of descending tips.

Details

The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child).

Tip labels and uncollapsed node labels of the collapsed tree are inheritted from the original tree. If `rename_collapsed_nodes==FALSE`, then labels of collapsed nodes will be the node labels from the original tree (in this case the original tree should include node labels). If `rename_collapsed_nodes==TRUE`, each collapsed node is given the label of its farthest descending tip. If `shorten==TRUE`, then edge lengths are the same as in the original tree. If `shorten==FALSE`, then edges leading into collapsed nodes may be longer than before.

Value

A list with the following elements:

`tree`	A new rooted tree of class "phylo", containing the collapsed tree.
`root_shift`	Numeric, indicating the phylogenetic distance between the old and the new root. Will always be non-negative.
`collapsed_nodes`	Integer vector, listing indices of collapsed nodes in the original tree (subset of 1,..,Nnodes).
`new2old_clade`	Integer vector of length equal to the number of tips+nodes in the collapsed tree, with values in 1,..,Ntips+Nnodes, mapping tip/node indices of the collapsed tree to tip/node indices in the original tree.
`new2old_edge`	Integer vector of length equal to the number of edges in the collapsed tree, with values in 1,..,Nedges, mapping edge indices of the collapsed tree to edge indices in the original tree.

Author(s)

Stilianos Louca

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=1000)$tree

# print number of nodes
cat(sprintf("Simulated tree has %d nodes\n",tree$Nnode))

# collapse any nodes with tip-distances < 20
collapsed = collapse_tree_at_resolution(tree, resolution=20)$tree

# print number of nodes
cat(sprintf("Collapsed tree has %d nodes\n",collapsed$Nnode))
```

`congruent_divergence_times` *Extract dating anchors for a target tree, using a dated reference tree*

Description

Given a reference tree and a target tree, this function maps target nodes to concordant reference nodes when possible, and extracts divergence times of the mapped reference nodes from the reference tree. This function can be used to define secondary dating constraints for a larger target tree, based on a time-calibrated smaller reference tree (Eastman et al. 2013). This only makes sense if the reference tree is time-calibrated. A provided mapping specifies which and how tips in the target tree correspond to tips in the reference tree.

Usage

```
congruent_divergence_times(reference_tree, target_tree, mapping)
```

Arguments

`reference_tree`	A rooted tree object of class "phylo". Usually this tree will be time-calibrated (i.e. edge lengths represent time intervals).
`target_tree`	A rooted tree object of class "phylo".
`mapping`	A table mapping a subset of target tips to a subset of reference tips, as described by Eastman et al (2013). Multiple target tips may map to the same reference tip, but not vice versa (i.e. every target tip can appear at most once in the mapping). In general, a tip mapped to in the reference tree is assumed to represent a monophyletic group of tips in the target tree, although this assumption may be violated in practice (Eastman et al. 2013). The `mapping` must be in one of the following formats: Option 1: A 2D integer array of size NM x 2 (with NM being the number of mapped target tips), listing target tip indices mapped to reference tip indices (mapping[m,1] (target tip) –> mapping[m,2] (reference tip)).

Option 2: A 2D character array of size NM x 2, listing target tip labels mapped to reference tip labels.

Option 3: A data frame of size NM x 1, whose row names are target tip labels and whose entries are either integers (reference tip indices) or characters (reference tip labels). This is the format used by `geiger::congruify.phylo` (v.206).

Option 4: A vector of size NM, whose names are target tip labels and whose entries are either integers (reference tip indices) or characters (reference tip labels).

Details

Both the reference and target tree may include monofurcations and/or multifurcations. In principle, neither of the two trees needs to be ultrametric, although in most applications `reference_tree` will be ultrametric.

In special cases each reference tip may be found in the target tree, i.e. the reference tree is a subtree of the target tree. This may occur e.g. if a smaller subtree of the target tree has been extracted and dated, and subsequently the larger target tree is to be dated using secondary constraints inferred from the dated subtree.

The function returns a table that maps a subset of target nodes to an equally sized subset of concordant reference nodes. Ages (divergence times) of the mapped reference nodes are extracted and associated with the concordant target nodes.

For bifurcating trees the average time complexity of this function is O(TNtips x log(RNtips) x NM), where TNtips and RNtips are the number of tips in the target and reference tree, respectively. This function is similar to `geiger::congruify.phylo` (v.206). For large trees, this function tends to be much faster than `geiger::congruify.phylo`.

Value

A named list with the following elements:

`Rnodes`	Integer vector of length NC (where NC is the number of concordant node pairs found) and with values in 1,..,RNnodes, listing indices of reference nodes that could be matched with (i.e. were concordant to) a target node. Entries in `Rnodes` will correspond to entries in `Tnodes` and `ages`.
`Tnodes`	Integer vector of length NC and with values in 1,..,TNnodes, listing indices of target nodes that could be matched with (i.e. were concordant to) a reference node. Entries in `Tnodes` will correspond to entries in `Rnodes` and `ages`.
`ages`	Numeric vector of length NC, listing divergence times (ages) of the reference nodes listed in `Rnodes`. These ages can be used as fixed anchors for time-calibrating the target tree using a separate program (such as `PATHd8`).

Author(s)

Stilianos Louca

References

J. M. Eastman, L. J. Harmon, D. C. Tank (2013). Congruification: support for time scaling large phylogenetic trees. Methods in Ecology and Evolution. 4:688-691.

See Also

`extend_tree_to_height`, `date_tree_red`, `get_tips_for_mrcas`, `tree_distance`

Examples

```
# generate random tree (target tree)
Ntips = 10000
tree  = castor::generate_random_tree(parameters=list(birth_rate_intercept=1), max_tips=Ntips

# extract random subtree (reference tree)
Nsubtips    = 10
subtips     = sample.int(n=Ntips,size=Nsubtips,replace=FALSE)
subtreeing  = castor::get_subtree_with_tips(tree, only_tips=subtips)
subtree     = subtreeing$subtree

# map subset of target tips to reference tips
mapping = matrix(c(subtreeing$new2old_tip,(1:Nsubtips)),ncol=2,byrow=FALSE)

# extract divergence times by congruification
congruification = congruent_divergence_times(subtree, tree, mapping)

cat("Concordant target nodes:\n")
print(congruification$target_nodes)

cat("Ages of concordant nodes:\n")
print(congruification$ages)
```

`count_lineages_through_time`

Count number of lineages through time.

Description

Given a rooted phylogenetic tree whose edge lengths represent time intervals, calculate the number of lineages represented in the tree at various time points (e.g., spanning from 0 to the maximum time of any tip). The root is interpreted as time 0, and the distance of any node or tip from the root is interpreted as time elapsed since the root. This function defines an equidistant sequence of time points, and counts how many edges "cross" each time point. Optionally, the slopes and relative slopes of the clade-counts-vs-time curve are also returned. The slopes and relative slopes are approximations for the species birth rate and the per-capita species birth rate (assuming no extinctions occurred).

Usage

```
count_lineages_through_time(  tree,
                              Ntimes         = NULL,
                              min_time       = NULL,
                              max_time       = NULL,
                              times          = NULL,
                              include_slopes= FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo", where edge lengths represent time intervals (or similar). The root is assumed to be the unique node with no incoming edge.
`Ntimes`	Integer, number of equidistant time points for which to calculade clade counts. Can also be `NULL`, in which case `times` must be provided.
`min_time`	Minimum time (distance from root) to consider. If `NULL`, this will be set to the minimum possible (i.e. 0). Only relevant if `times==NULL`.
`max_time`	Maximum time (distance from root) to consider. If `NULL`, this will be set to the maximum possible. Only relevant if `times==NULL`.
`times`	Integer vector, listing time points (in ascending order) for which to calculate clade counts. Can also be `NULL`, in which case `Ntimes` must be provided.
`include_slopes`	Logical, specifying whether the slope and the relative slope of the returned clades-per-time-point curve should also be returned.

Details

If `tree$edge.length` is missing, then every edge in the tree is assumed to be of length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The tree need not be ultrametric, although in general this function only makes sense for dated trees (e.g., where edge lengths are time intervals or similar).

Either `Ntimes` or `times` must be non-`NULL`, but not both. If `times!=NULL`, then `min_time` and `max_time` must be `NULL`.

Value

A list with the following elements:

`Ntimes`	Integer, indicating the number of returned time points. Equal to the provided `Ntimes` if applicable.
`times`	Numeric vector of size Ntimes, listing the considered time points in increasing order. If `times` was provided as an argument to the function, then this will be the same as provided.
`lineages`	Integer vector of size Ntimes, listing the number of lineages represented in the tree at each time point.
`slopes`	Numeric vector of size Ntimes, listing the slopes (finite-difference approximation of 1st derivative) of the curve clade_counts vs time_point.

`relative_slopes`	Numeric vector of size Ntimes, listing the relative slopes of the curve clade_counts vs time_point, i.e. `slopes` divided by a sliding-window average of `clade_counts`.

Author(s)

Stilianos Louca

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1), max_tips=1000)$tree

# count clades over time
results = count_lineages_through_time(tree, Ntimes=100)

# plot curve (number of clades vs time)
plot(results$times, results$lineages, type="l", xlab="time", ylab="# clades")
```

`count_tips_per_node` *Count descending tips.*

Description

Given a rooted phylogenetic tree, count the number of tips descending (directy or indirectly) from each node.

Usage

```
count_tips_per_node(tree)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.

Details

The asymptotic time complexity of this function is O(Nedges), where Nedges is the number of edges.

Value

An integer vector of size Nnodes, with the i-th entry being the number of tips descending (directly or indirectly) from the i-th node.

Author(s)

Stilianos Louca

See Also

`get_subtree_at_node`

Examples

```
# generate a tree using a simple speciation model
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=1000)$tree

# count number of tips descending from each node
tips_per_node = count_tips_per_node(tree);

# plot histogram of tips-per-node
barplot(table(tips_per_node[tips_per_node<10]), xlab="# tips", ylab="# nodes")
```

`date_tree_red` *Date a tree based on relative evolutionary divergences.*

Description

Given a rooted phylogenetic tree and a single node ('anchor') of known age (distance from the present), rescale all edge lengths so that the tree becomes ultrametric and edge lengths correspond to time intervals. The function is based on relative evolutionary divergences (RED), which measure the relative position of each node between the root and its descending tips (Parks et al. 2018). If no anchor node is provided, the root is simply assumed to have age 1. This function provides a heuristic quick-and-dirty way to date a phylogenetic tree.

Usage

```
date_tree_red(tree, anchor_node = NULL, anchor_age = 1)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`anchor_node`	Integer, ranging between 1 and Nnodes. Index of the node to be used as dating anchor. If `NULL`, the tree's root is used as anchor.
`anchor_age`	Positive numeric. Age of the anchor node.

Details

The RED of a node measures its relative placement between the root and the node's descending tips (Parks et al. 2018). The root's RED is set to 0. Traversing from root to tips (preorder traversal), for each node the RED is set to $P + (a/(a+b)) \cdot (1-P)$, where P is the RED of the node's parent, a is the edge length connecting the node to its parent, and b is the average distance from the node to its descending tips. The RED of all tips is set to 1.

For each edge, the RED difference between child & parent is used to set the new length of that edge, multiplied by some common scaling factor to translate RED units into time units. The scaling factor is chosen such that the new distance of the anchor node from its descending tips equals `anchor_age`. All tips will have age 0. The topology of the dated tree, as well as tip/node/edge indices, remain unchanged.

This function provides a heuristic approach to making a tree ultrametric, and has not been derived from a specific evolutionary model. In particular, its statistical properties are unknown to the author.

The time complexity of this function is O(Nedges). The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). If `tree$edge.length` is `NULL`, then all edges in the input tree are assumed to have length 1.

Value

A list with the following elements:

`success`	Logical, indicating whether the dating was successful. If `FALSE`, all other return values (except for `error`) may be undefined.
`tree`	A new rooted tree of class "phylo", representing the dated tree.
`REDs`	Numeric vector of size Nnodes, listing the RED of each node in the input tree.
`error`	Character, listing any error message if `success==FALSE`.

Author(s)

Stilianos Louca

References

D. H. Parks, M. Chuvochina et al. (2018). A proposal for a standardized bacterial taxonomy based on genome phylogeny. bioRxiv 256800. DOI:10.1101/256800

See Also

`congruent_divergence_times`

Examples

```
# generate a random non-ultrametric tree
params = list(birth_rate_intercept=1, death_rate_intercept=0.8)
tree = generate_random_tree(params, max_time=1000, coalescent=FALSE)$tree

# make ultrametric, by setting the root to 2 million years
dated_tree = date_tree_red(tree, anchor_age=2e6)
```

`exponentiate_matrix` *Exponentiate a matrix.*

Description

Calculate the exponential $\exp(T \cdot A)$ of some quadratic real-valued matrix A for one or more scalar scaling factors T.

Usage

```
exponentiate_matrix(A, scalings=1, max_absolute_error=1e-3,
                    min_polynomials=1, max_polynomials=1000)
```

Arguments

`A`	A real-valued quadratic matrix of size N x N.
`scalings`	Vector of real-valued scalar scaling factors T, for each of which the exponential $\exp(T \cdot A)$ should be calculated.
`max_absolute_error`	Maximum allowed absolute error for the returned approximations. A smaller allowed error implies a greater computational cost as more matrix polynomials need to be included (see below). The returned approximations may have a greater error if the parameter `max_polynomials` is set too low.
`min_polynomials`	Minimum number of polynomials to include in the approximations (see equation below), even if `max_absolute_error` may be satisfied with fewer polynomials. If you don't know how to choose this, in most cases the default is fine. Note that regardless of `min_polynomials` and `max_absolute_error`, the number of polynomials used will not exceed `max_polynomials`.
`max_polynomials`	Maximum allowed number of polynomials to include in the approximations (see equation below). Meant to provide a safety limit for the amount of memory and the computation time required. For example, a value of 1000 means that up to 1000 matrices (powers of A) of size N x N may be computed and stored temporarily in memory. Note that if `max_polynomials` is too low, the requested accuracy (via `max_absolute_error`) may not be achieved. That said, for large trees more memory may be required to store the actual result rather than the intermediate polynomials, i.e. for purposes of saving RAM it doesn't make much sense to choose `max_polynomials` much smaller than the length of `scalings`.

Details

Discrete character evolution Markov models often involve repeated exponentiations of the same transition matrix along each edge of the tree (i.e. with the scaling T being the edge length). Matrix exponentiation can become a serious computational bottleneck for larger trees or large matrices (i.e. spanning multiple discrete states). This function pre-calculates polynomials $A^p/p!$ of the matrix, and then uses linear combinations of the same polynomials for each requested T:

$$\exp(T \cdot A) = \sum_{p=0}^{P} T^p \frac{A^p}{p!} + ...$$

This function thus becomes very efficient when the number of scaling factors is large (e.g. >10,000). The number of polynomials included is determined based on the specified `max_absolute_error`, and based on the largest (by magnitude) scaling factor requested. The function utilizes the balancing algorithm proposed by James et al (2014, Algorithm 3) and the scaling & squaring method (Moler and Van Loan, 2003) to improve the conditioning of the matrix prior to exponentiation.

Value

A 3D numeric matrix of size N x N x S, where N is the number of rows & column of the input matrix A and S is the length of `scalings`. The [r,c,s]-th element of this matrix is the entry in the r-th row and c-th column of $\exp(scalings[s] \cdot A)$.

Author(s)

Stilianos Louca

References

R. James, J. Langou and B. R. Lowery (2014). On matrix balancing and eigenvector computation. arXiv:1401.5766

C. Moler and C. Van Loan (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Review. 45:3-49.

Examples

```
# create a random 5 x 5 matrix
A = get_random_mk_transition_matrix(Nstates=5, rate_model="ER")

# calculate exponentials exp(0.1*A) and exp(10*A)
exponentials = exponentiate_matrix(A, scalings=c(0.1,10))

# print 1st exponential: exp(0.1*A)
print(exponentials[,,1])

# print 2nd exponential: exp(10*A)
print(exponentials[,,2])
```

`extend_tree_to_height` *Extend a rooted tree up to a specific height.*

Description

Given a rooted phylogenetic tree and a specific distance from the root ("new height"), elongate terminal edges (i.e. leading into tips) as needed so that all tips have a distance from the root equal to the new height. If a tip already extends beyond the specified new height, its incoming edge remains unchanged.

Usage

```
extend_tree_to_height(tree, new_height=NULL)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`new_height`	Numeric, specifying the phylogenetic distance from the root to which tips are to be extended. If `NULL` or negative, then it is set to the maximum distance of any tip from the root.

Details

The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). All tip, edge and node indices remain unchanged. This function provides a quick-and-dirty way to make a tree ultrametric, or to correct small numerical inaccuracies in supposed-to-be ultrametric trees.

Value

A list with the following elements:

`tree`	A new rooted tree of class "phylo", representing the extended tree.
`max_extension`	Numeric. The largest elongation added to a terminal edge.

Author(s)

Stilianos Louca

See Also

`trim_tree_at_height`

Examples

```
# generate a random non-ultrametric tree
tree = generate_random_tree(list(birth_rate_intercept=1,death_rate_intercept=0.5),
                            max_time=1000,
                            coalescent=FALSE)$tree

# print min & max distance from root
span = get_tree_span(tree)
cat(sprintf("Min & max tip height = %g & %g\n",span$min_distance,span$max_distance))

# make tree ultrametric by extending terminal edges
extended = extend_tree_to_height(tree)$tree

# print new min & max distance from root
span = get_tree_span(extended)
cat(sprintf("Min & max tip height = %g & %g\n",span$min_distance,span$max_distance))
```

---

find_farthest_tips *Find farthest tip to each tip & node of a tree.*

Description

Given a rooted phylogenetic tree and a subset of potential target tips, for each tip and node in the tree find the farthest target tip. The set of target tips can also be taken as the whole set of tips in the tree.

Usage

```
find_farthest_tips( tree,
                    only_descending_tips = FALSE,
                    target_tips          = NULL,
                    as_edge_counts       = FALSE,
                    check_input          = TRUE)
```

Arguments

`tree` A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.

`only_descending_tips`
A logical indicating whether the farthest tip to a node or tip should be chosen from its descending tips only. If FALSE, then the whole set of possible target tips is considered.

`target_tips` Optional integer vector or character vector listing the subset of target tips to restrict the search to. If an integer vector, this should list tip indices (values in 1,..,Ntips). If a character vector, it should list tip names (in this case `tree$tip.label` must exist). If `target_tips` is `NULL`, then all tips of the tree are considered as target tips.

`as_edge_counts`
: Logical, specifying whether to count phylogenetic distance in terms of edge counts instead of cumulative edge lengths. This is the same as setting all edge lengths to 1.

`check_input`
: Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.

Details

If `only_descending_tips` is `TRUE`, then only descending target tips are considered when searching for the farthest target tip of a node/tip. In that case, if a node/tip has no descending target tip, its farthest target tip is set to NA. If `tree$edge.length` is missing or NULL, then each edge is assumed to have length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child).

The asymptotic time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

A list with the following elements:

`farthest_tip_per_tip`
: An integer vector of size Ntips, listing the farthest target tip for each tip in the tree. Hence, `farthest_tip_per_tip[i]` is the index of the farthest tip (from the set of target tips), with respect to tip i (where i=1,..,Ntips). Some values may appear multiple times in this vector, if multiple tips share the same farthest target tip.

`farthest_tip_per_node`
: An integer vector of size Nnodes, listing the index of the farthest target tip for each node in the tree. Hence, `farthest_tip_per_node[i]` is the index of the farthest tip (from the set of target tips), with respect to node i (where i=1,..,Nnodes). Some values may appear multiple times in this vector, if multiple nodes share the same farthest target tip.

`farthest_distance_per_tip`
: Integer vector of size Ntips. Phylogenetic ("patristic") distance of each tip in the tree to its farthest target tip. If `only_descending_tips` was set to `TRUE`, then `farthest_distance_per_tip[i]` will be set to infinity for any tip i that is not a target tip.

`farthest_distance_per_node`
: Integer vector of size Nnodes. Phylogenetic ("patristic") distance of each node in the tree to its farthest target tip. If `only_descending_tips` was set to `TRUE`, then `farthest_distance_per_node[i]` will be set to infinity for any node i that has no descending target tips.

Author(s)

Stilianos Louca

References

M. G. I. Langille, J. Zaneveld, J. G. Caporaso et al (2013). Predictive functional profiling of microbial communities using 16S rRNA marker gene sequences. Nature Biotechnology. 31:814-821.

See Also

find_nearest_tips

Examples

```
# generate a random tree
Ntips = 1000
parameters = list(birth_rate_intercept=1,death_rate_intercept=0.9)
tree = generate_random_tree(parameters,Ntips,coalescent=FALSE)$tree

# pick a random set of "target" tips
target_tips = sample.int(n=Ntips, size=5, replace=FALSE)

# find farthest target tip to each tip & node in the tree
results = find_farthest_tips(tree, target_tips=target_tips)

# plot histogram of distances to target tips (across all tips of the tree)
distances = results$farthest_distance_per_tip
hist(distances, breaks=10, xlab="farthest distance", ylab="number of tips", prob=FALSE);
```

`find_nearest_tips` *Find nearest tip to each tip & node of a tree.*

Description

Given a rooted phylogenetic tree and a subset of potential target tips, for each tip and node in the tree find the nearest target tip. The set of target tips can also be taken as the whole set of tips in the tree.

Usage

```
find_nearest_tips(tree,
                  only_descending_tips = FALSE,
                  target_tips          = NULL,
                  as_edge_counts       = FALSE,
                  check_input          = TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.

`only_descending_tips`
: A logical indicating whether the nearest tip to a node or tip should be chosen from its descending tips only. If FALSE, then the whole set of possible target tips is considered.

`target_tips`
: Optional integer vector or character vector listing the subset of target tips to restrict the search to. If an integer vector, this should list tip indices (values in 1,..,Ntips). If a character vector, it should list tip names (in this case `tree$tip.label` must exist). If `target_tips` is `NULL`, then all tips of the tree are considered as target tips.

`as_edge_counts`
: Logical, specifying whether to count phylogenetic distance in terms of edge counts instead of cumulative edge lengths. This is the same as setting all edge lengths to 1.

`check_input`
: Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.

Details

Langille et al. (2013) introduced the Nearest Sequenced Taxon Index (NSTI) as a measure for how well a set of microbial operational taxonomic units (OTUs) is represented by a set of sequenced genomes of related organisms. Specifically, the NSTI of a microbial community is the average phylogenetic distance of any OTU in the community, to the closest relative with an available sequenced genome ("target tips"). In analogy to the NSTI, the function `find_nearest_tips` provides a means to find the nearest tip (from a subset of target tips) to each tip and node in a phylogenetic tree, together with the corresponding phylogenetic ("patristic") distance.

If `only_descending_tips` is `TRUE`, then only descending target tips are considered when searching for the nearest target tip of a node/tip. In that case, if a node/tip has no descending target tip, its nearest target tip is set to NA. If `tree$edge.length` is missing or NULL, then each edge is assumed to have length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child).

The asymptotic time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

A list with the following elements:

`nearest_tip_per_tip`
: An integer vector of size Ntips, listing the nearest target tip for each tip in the tree. Hence, `nearest_tip_per_tip[i]` is the index of the nearest tip (from the set of target tips), with respect to tip i (where i=1,..,Ntips). Some values may appear multiple times in this vector, if multiple tips share the same nearest target tip.

`nearest_tip_per_node`
: An integer vector of size Nnodes, listing the index of the nearest target tip for each node in the tree. Hence, `nearest_tip_per_node[i]` is the index of the nearest tip (from the set of target tips), with respect to node i (where

i=1,..,Nnodes). Some values may appear multiple times in this vector, if multiple nodes share the same nearest target tip.

`nearest_distance_per_tip`
: Integer vector of size Ntips. Phylogenetic ("patristic") distance of each tip in the tree to its nearest target tip. If `only_descending_tips` was set to `TRUE`, then `nearest_distance_per_tip[i]` will be set to infinity for any tip i that is not a target tip.

`nearest_distance_per_node`
: Integer vector of size Nnodes. Phylogenetic ("patristic") distance of each node in the tree to its nearest target tip. If `only_descending_tips` was set to `TRUE`, then `nearest_distance_per_node[i]` will be set to infinity for any node i that has no descending target tips.

Author(s)

Stilianos Louca

References

M. G. I. Langille, J. Zaneveld, J. G. Caporaso et al (2013). Predictive functional profiling of microbial communities using 16S rRNA marker gene sequences. Nature Biotechnology. 31:814-821.

See Also

`find_farthest_tips`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# pick a random set of "target" tips
target_tips = sample.int(n=Ntips, size=as.integer(Ntips/10), replace=FALSE)

# find nearest target tip to each tip & node in the tree
results = find_nearest_tips(tree, target_tips=target_tips)

# plot histogram of distances to target tips (across all tips of the tree)
distances = results$nearest_distance_per_tip
hist(distances, breaks=10, xlab="nearest distance", ylab="number of tips", prob=FALSE);
```

`find_root` *Find the root of a tree.*

Description

Find the root of a phylogenetic tree. The root is defined as the unique node with no parent.

Usage

```
find_root(tree)
```

Arguments

`tree`	A tree of class "phylo". If the tree is not rooted, the function will return `NA`.

Details

By convention, the root of a "phylo" tree is typically the first node (i.e. with index Ntips+1), however this is not always guaranteed. This function finds the root of a tree by searching for the node with no parent. If no such node exists, `NA` is returned. If multiple such nodes exist, `NA` is returned. If any node has more than 1 parent, `NA` is returned. Hence, this function can be used to test if a tree is rooted purely based on the edge structure, assuming that the tree is connected (i.e. not a forest).

The asymptotic time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

Index of the tree's root, as listed in `tree$edge`. An integer ranging from Ntips+1 to Ntips+Nnodes, where Ntips and Nnodes is the number of tips and nodes in the tree, respectively. By convention, the root will typically be Ntips+1 but this is not guaranteed.

Author(s)

Stilianos Louca

See Also

`find_root_of_monophyletic_tips`, `root_at_node`, `root_at_midpoint`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# reroot the tree at the 20-th node
new_root_node = 20
tree = root_at_node(tree, new_root_node, update_indices=FALSE)

# find new root index and compare with expectation
cat(sprintf("New root is %d, expected at %d\n",find_root(tree),new_root_node+Ntips))
```

find_root_of_monophyletic_tips

Find the node or tip that, as root, would make a set of target tips monophyletic.

Description

Given a tree (rooted or unrooted) and a specific set of target tips, this function finds the tip or node that, if turned into root, would make a set of target tips a monophyletic group that either descends from a single child of the new root (if `as_MRCA==FALSE`) or whose MRCA is the new root (if `as_MRCA==TRUE`).

Usage

```
find_root_of_monophyletic_tips(tree, monophyletic_tips, as_MRCA=TRUE, is_rooted=FAI
```

Arguments

`tree`	A tree object of class "phylo". Can be unrooted or rooted.
`monophyletic_tips`	Character or integer vector, specifying the names or indices, respectively, of the target tips that should be turned monophyletic. If an integer vector, its elements must be between 1 and Ntips. If a character vector, its elements must be elements in `tree$tip.label`.
`as_MRCA`	Logical, specifying whether the new root should become the MRCA of the target tips. If `FALSE`, the new root is chosen such that the MRCA of the target tips is the child of the new root.
`is_rooted`	Logical, specifying whether the input tree can be assumed to be rooted. If you are sure that the input tree is rooted, set this to `TRUE` for computational efficiency, otherwise to be on the safe side set this to `FALSE`.

Details

The input tree may include an arbitrary number of incoming and outgoing edges per node (but only one edge per tip), and the direction of these edges can be arbitrary. Of course, the undirected graph defined by all edges must still be a valid tree (i.e. a connected acyclic graph). Note that this function does not change the tree, it just determines which tip or node should be made root for the target tips to be a monophyletic group.

The asymptotic time complexity of this function is O(Nedges).

Value

A single integer between 1 and (Ntips+Nnodes), specifying the index of the tip or node that, if made root, would make the target tips monophyletic. If this was not possible, `NA` is returned.

Author(s)

Stilianos Louca

See Also

`find_root`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# pick a random node and find all descending tips
MRCA = sample.int(tree$Nnode,size=1)
monophyletic_tips = get_subtree_at_node(tree, MRCA)$new2old_tip

# change root of tree (change edge directions)
tree = root_at_node(tree, new_root_node=10, update_indices=FALSE)

# determine root that would make target tips monophyletic
new_root = find_root_of_monophyletic_tips(tree, monophyletic_tips, as_MRCA=TRUE, is_rooted=F

# compare expectation with result
cat(sprintf("MRCA = %d, new root node=%d\n",MRCA,new_root-Ntips))
```

`fit_bm_model` *Fit a Brownian motion model for multivariate trait evolution.*

Description

Given a rooted phylogenetic tree and states of one or more continuous (numeric) traits on the tree's tips, fit a multivariate Brownian motion model of correlated co-evolution of these traits. This estimates a single diffusivity matrix, which describes the variance-covariance structure of each trait's random walk. The model assumes a fixed diffusivity matrix on the entire tree.

Usage

```
fit_bm_model(tree, tip_states, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, or a 2D numeric matrix of size Ntips x Ntraits, specifying the numeric state of each trait at each tip in the tree.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

The BM model is defined by the stochastic differential equation

$$dX = \sigma \cdot dW$$

where W is a multidimensional Wiener process with Ndegrees independent components and σ is a matrix of size Ntraits x Ndegrees. Alternatively, the same model can be defined as a Fokker-Planck equation for the probability density ρ:

$$\frac{\partial \rho}{\partial t} = \sum_{i,j} D_{ij} \frac{\partial^2 \rho}{\partial x_i \partial x_j}.$$

The matrix D is referred to as the diffusivity matrix (or diffusion tensor), and $2D = \sigma \cdot \sigma^T$. Note that σ can be obtained from D by means of a Cholesky decomposition.

The function uses phylogenetic independent contrasts (Felsenstein, 1985) to retrieve independent increments of the multivariate random walk. The diffusivity matrix D is then fitted using maximum-likelihood on the intrinsic geometry of positive-definite matrices.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multifurcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child). Note that multifurcations are internally expanded to bifurcations, prior to model fitting.

Value

A list with the following elements:

`diffusivity`	Either a single non-negative number (if `tip_states` was a vector) or a 2D quadratic non-negative-definite matrix (if `tip_states` was a 2D matrix). The fitted diffusivity matrix of the multivariate Brownian motion model.
`loglikelihood`	The log-likelihood of the fitted model, given the provided tip states data.

Author(s)

Stilianos Louca

References

J. Felsenstein (1985). Phylogenies and the Comparative Method. The American Naturalist. 125:1-15.

See Also

`simulate_bm_model`, `get_independent_contrasts`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1), 10000)$tree


# Example 1: Scalar case
# - - - - - - - - - - - - - - -
# simulate scalar continuous trait on the tree
D = 1
tip_states = simulate_bm_model(tree, diffusivity=D)$tip_states

# estimate original diffusivity from the generated data
fit = fit_bm_model(tree, tip_states)
cat(sprintf("True D=%g, fitted D=%g\n",D,fit$diffusivity))


# Example 2: Multivariate case
# - - - - - - - - - - - - - - - -
# simulate vector-valued continuous trait on the tree
D = get_random_diffusivity_matrix(Ntraits=5)
tip_states = simulate_bm_model(tree, diffusivity=D)$tip_states

# estimate original diffusivity matrix from the generated data
fit = fit_bm_model(tree, tip_states)

# compare true and fitted diffusivity matrices
cat("True D:\n"); print(D)
cat("Fitted D:\n"); print(fit$diffusivity)
```

`fit_hbd_model_on_grid` *Fit a homogenous birth-death model on a discrete time grid.*

Description

Given an ultrametric timetree, fit a homogenous birth-death (HBD) model in which speciation and extinction rates (λ and mu) are defined on a fixed grid of discrete time points and assumed to vary polynomially between grid points. “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates (in the literature this is sometimes referred to simply as “birth-death model”). Every HBD model is defined based on the values that λ and μ take over time as well as the sampling fraction ρ (fraction of extant species sampled). This function estimates the values of λ and μ at each grid point by maximizing the likelihood (Morlon et al. 2011) of the timetree under the resulting HBD model.

Usage

```
fit_hbd_model_on_grid(tree,
                         oldest_age        = NULL,
                         age_grid          = NULL,
                         min_lambda        = 0,
                         max_lambda        = +Inf,
                         min_mu            = 0,
                         max_mu            = +Inf,
                         min_rho           = 1e-10,
                         max_rho           = 1,
                         guess_lambda      = NULL,
                         guess_mu          = NULL,
                         guess_rho         = 1,
                         fixed_lambda      = NULL,
                         fixed_mu          = NULL,
                         fixed_rho         = NULL,
                         const_lambda      = FALSE,
                         const_mu          = FALSE,
                         splines_degree    = 1,
                         condition         = "stem",
                         relative_dt       = 1e-3,
                         Ntrials           = 1,
                         Nthreads          = 1,
                         max_model_runtime = NULL,
                         fit_control       = list())
```

Arguments

`tree`	An ultrametric timetree of class "phylo", representing the time-calibrated reconstructed phylogeny of a set of extant species.
`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to consider when calculating the likelihood. If this is equal to or greater than the root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.
`age_grid`	Numeric vector, listing ages in ascending order, on which λ and μ are allowed to vary independently. This grid must cover at least the age range from 0 to `oldest_age`. If `NULL` or of length <=1 (regardless of value), then λ and μ are assumed to be time-independent.
`min_lambda`	Numeric vector of length Ngrid (=`max(1,length(age_grid))`), or a single numeric, specifying lower bounds for the fitted λ at each point in the age grid. If a single numeric, the same lower bound applies at all ages.
`max_lambda`	Numeric vector of length Ngrid, or a single numeric, specifying upper bounds for the fitted λ at each point in the age grid. If a single numeric, the same upper bound applies at all ages. Use `+Inf` to omit upper bounds.
`min_mu`	Numeric vector of length Ngrid, or a single numeric, specifying lower bounds

	for the fitted μ at each point in the age grid. If a single numeric, the same lower bound applies at all ages.
`max_mu`	Numeric vector of length Ngrid, or a single numeric, specifying upper bounds for the fitted μ at each point in the age grid. If a single numeric, the same upper bound applies at all ages. Use `+Inf` to omit upper bounds.
`min_rho`	Numeric, specifying a lower bound for the fitted sampling fraction ρ (fraction of extant species included in the tree).
`max_rho`	Numeric, specifying an upper bound for the fitted sampling fraction ρ.
`guess_lambda`	Initial guess for λ at each age-grid point. Either `NULL` (an initial guess will be computed automatically), or a single numeric (guessing the same λ at all ages) or a numeric vector of size Ngrid specifying a separate guess for λ at each age-grid point. To omit an initial guess for some but not all age-grid points, set their guess values to `NA`. Guess values are ignored for non-fitted (i.e., fixed) parameters.
`guess_mu`	Initial guess for μ at each age-grid point. Either `NULL` (an initial guess will be computed automatically), or a single numeric (guessing the same μ at all ages) or a numeric vector of size Ngrid specifying a separate guess for μ at each age-grid point. To omit an initial guess for some but not all age-grid points, set their guess values to `NA`. Guess values are ignored for non-fitted (i.e., fixed) parameters.
`guess_rho`	Numeric, specifying an initial guess for the sampling fraction ρ. Setting this to `NULL` or `NA` is the same as setting it to 1.
`fixed_lambda`	Optional fixed (i.e. non-fitted) λ values on one or more age-grid points. Either `NULL` (λ is not fixed anywhere), or a single numeric (λ fixed to the same value at all grid points) or a numeric vector of size Ngrid (λ fixed on one or more age-grid points, use `NA` for non-fixed values).
`fixed_mu`	Optional fixed (i.e. non-fitted) μ values on one or more age-grid points. Either `NULL` (μ is not fixed anywhere), or a single numeric (μ fixed to the same value at all grid points) or a numeric vector of size Ngrid (μ fixed on one or more age-grid points, use `NA` for non-fixed values).
`fixed_rho`	Numeric between 0 and 1, optionallly specifying a fixed value for the sampling fraction ρ. If `NULL` or `NA`, the sampling fraction ρ is estimated, however note that this may not always be meaningful (Stadler 2009, Stadler 2013).
`const_lambda`	Logical, specifying whether λ should be assumed constant across the grid, i.e. time-independent. Setting `const_lambda=TRUE` reduces the number of free (i.e., independently fitted) parameters. If λ is fixed on some grid points (i.e. via `fixed_lambda`), then only the non-fixed lambdas are assumed to be identical to one another.
`const_mu`	Logical, specifying whether μ should be assumed constant across the grid, i.e. time-independent. Setting `const_mu=TRUE` reduces the number of free (i.e., independently fitted) parameters. If μ is fixed on some grid points (i.e. via `fixed_mu`), then only the non-fixed mus are assumed to be identical to one another.
`splines_degree`	Integer between 0 and 3 (inclusive), specifying the polynomial degree of λ and μ between age-grid points. If 0, then λ and μ are considered piecewise constant,

if 1 then λ and μ are considered piecewise linear, if 2 or 3 then λ and μ are considered to be splines of degree 2 or 3, respectively. The `splines_degree` influences the analytical properties of the curve, e.g. `splines_degree==1` guarantees a continuous curve, `splines_degree==2` guarantees a continuous curve and continuous derivative, and so on. A degree of 0 is generally not recommended.

`condition` Character, either "crown", "stem" or "none", specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age. "none" is generally not recommended.

`relative_dt` Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time, when calculating the likelihood. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

`Ntrials` Integer, specifying the number of independent fitting trials to perform, each starting from a random choice of model parameters. Increasing `Ntrials` reduces the risk of reaching a non-global local maximum in the fitting objective.

`Nthreads` Integer, specifying the number of parallel threads to use for performing multiple fitting trials simultaneously. This should generally not exceed the number of available CPUs on your machine. Parallel computing is not available on the Windows platform.

`max_model_runtime`
Optional numeric, specifying the maximum number of seconds to allow for each evaluation of the likelihood function. Use this to abort fitting trials leading to parameter regions where the likelihood takes a long time to evaluate (these are often unlikely parameter regions).

`fit_control` Named list containing options for the `nlminb` optimization routine, such as `iter.max`, `eval.max` or `rel.tol`. For a complete list of options and default values see the documentation of `nlminb` in the `stats` package.

Details

It is generally advised to provide as much information to the function `fit_hbd_model_on_grid` as possible, including reasonable lower and upper bounds (`min_lambda`, `max_lambda`, `min_mu`, `max_mu`, `min_rho` and `max_rho`) and a reasonable parameter guess (`guess_lambda`, `guess_mu` and `guess_rho`). It is also important that the `age_grid` is sufficiently fine to capture the expected major variations of λ and μ over time, but keep in mind the serious risk of overfitting when `age_grid` is too fine and/or the tree is too small.

Value

A list with the following elements:

`success` Logical, indicating whether model fitting succeeded. If `FALSE`, the returned list will include an additional “error” element (character) providing a description of the error; in that case all other return variables may be undefined.

`objective_value`	The maximized fitting objective. Currently, only maximum-likelihood estimation is implemented, and hence this will always be the maximized log-likelihood.
`objective_name`	The name of the objective that was maximized during fitting. Currently, only maximum-likelihood estimation is implemented, and hence this will always be "loglikelihood".
`loglikelihood`	The log-likelihood of the fitted model for the given timetree.
`fitted_lambda`	Numeric vector of size Ngrid, listing fitted or fixed speciation rates λ at each age-grid point.
`fitted_mu`	Numeric vector of size Ngrid, listing fitted or fixed extinction rates μ at each age-grid point.
`fitted_rho`	Numeric, specifying the fitted or fixed sampling fraction ρ.
`guess_lambda`	Numeric vector of size Ngrid, specifying the initial guess for λ at each age-grid point.
`guess_mu`	Numeric vector of size Ngrid, specifying the initial guess for μ at each age-grid point.
`guess_rho`	Numeric, specifying the initial guess for ρ.
`age_grid`	The age-grid on which λ and μ are defined. This will be the same as the provided `age_grid`, unless the latter was `NULL` or of length <=1.
`NFP`	Integer, number of free (i.e., independently) fitted parameters. If none of the λ, μ and ρ were fixed, and `const_lambda=FALSE` and `const_mu=FALSE`, then `NFP` will be equal to 2*Ngrid+1.
`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k - 2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`converged`	Logical, specifying whether the maximum likelihood was reached after convergence of the optimization algorithm. Note that in some cases the maximum likelihood may have been achieved by an optimization path that did not yet converge (in which case it's advisable to increase `iter.max` and/or `eval.max`).
`Niterations`	Integer, specifying the number of iterations performed during the optimization path that yielded the maximum likelihood.
`Nevaluations`	Integer, specifying the number of likelihood evaluations performed during the optimization path that yielded the maximum likelihood.

Author(s)

Stilianos Louca

References

T. Stadler (2009). On incomplete sampling under birth-death models and connections to the sampling-based coalescent. Journal of Theoretical Biology. 261:58-66.

T. Stadler (2013). How can we improve accuracy of macroevolutionary rate estimates? Systematic Biology. 62:321-329.

H. Morlon, T. L. Parsons, J. B. Plotkin (2011). Reconciling molecular phylogenies with the fossil record. Proceedings of the National Academy of Sciences. 108:16327-16332.

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`simulate_deterministic_hbd`

`loglikelihood_hbd`

`fit_hbd_model_parametric`

`fit_hbd_pdr_on_grid`

`fit_hbd_pdr_parametric`

Examples

```
## Not run:
# Generate a random tree with exponentially varying lambda & mu
Ntips     = 10000
rho       = 0.5 # sampling fraction
time_grid = seq(from=0, to=100, by=0.01)
lambdas   = 2*exp(0.1*time_grid)
mus       = 1.5*exp(0.09*time_grid)
sim       = generate_random_tree( parameters  = list(rarefaction=rho),
                                  max_tips    = Ntips/rho,
                                  coalescent  = TRUE,
                                  added_rates_times     = time_grid,
                                  added_birth_rates_pc  = lambdas,
                                  added_death_rates_pc  = mus)
tree = sim$tree
root_age = castor::get_tree_span(tree)$max_distance
cat(sprintf("Tree has %d tips, spans %g Myr\n",length(tree$tip.label),root_age))


# Fit mu on grid
# Assume that lambda & rho are known
Ngrid     = 10
age_grid  = seq(from=0,to=root_age,length.out=Ngrid)
fit = fit_hbd_model_on_grid(tree,
          age_grid    = age_grid,
          max_mu      = 100,
          fixed_lambda= approx(x=time_grid,y=lambdas,xout=sim$final_time-age_grid)$y,
          fixed_rho   = rho,
          condition   = "crown",
          Ntrials     = 10,# perform 10 fitting trials
          Nthreads    = 10,# use two CPUs
          max_model_runtime = 1)  # limit model evaluation to 1 second
if(!fit$success){
  cat(sprintf("ERROR: Fitting failed: %s\n",fit$error))
}else{
  cat(sprintf("Fitting succeeded:\nLoglikelihood=%g\n",fit$loglikelihood))
  # plot fitted & true mu
  plot( x     = fit$age_grid,
        y     = fit$fitted_mu,
        main  = 'Fitted & true mu',
        xlab  = 'age',
        ylab  = 'mu',
        type  = 'b',
        col   = 'red',
        xlim  = c(root_age,0))
  lines(x     = sim$final_time-time_grid,
        y     = mus,
        type  = 'l',
        col   = 'blue');
}

## End(Not run)
```

`fit_hbd_model_parametric`
Fit a parametric homogenous birth-death model to a timetree.

Description

Given an ultrametric timetree, fit a homogenous birth-death (HBD) model in which speciation and extinction rates (λ and mu) are given as parameterized functions of time before present. “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates (in the literature this is sometimes referred to simply as “birth-death model”). Every HBD model is defined based on the values that λ and μ take over time as well as the sampling fraction ρ (fraction of extant species sampled); in turn, λ, μ and ρ can be parameterized by a finite set of parameters. This function estimates these parameters by maximizing the likelihood (Morlon et al. 2011) of the timetree under the resulting HBD model.

Usage

```
fit_hbd_model_parametric( tree,
                          param_values,
                          param_guess       = NULL,
                          param_min         = -Inf,
                          param_max         = +Inf,
                          param_scale       = NULL,
                          oldest_age        = NULL,
                          lambda,
                          mu,
                          rho,
                          age_grid          = NULL,
                          condition         = "stem",
                          relative_dt       = 1e-3,
                          Ntrials           = 1,
                          Nthreads          = 1,
                          max_model_runtime = NULL,
                          fit_control       = list())
```

Arguments

`tree`	An ultrametric timetree of class "phylo", representing the time-calibrated reconstructed phylogeny of a set of extant species.
`param_values`	Numeric vector, specifying fixed values for a some or all model parameters. For fitted (i.e., non-fixed) parameters, use `NaN` or `NA`. For example, the vector `c(1.5,NA,40)` specifies that the 1st and 3rd model parameters are fixed at the values 1.5 and 40, respectively, while the 2nd parameter is to be fitted. The length of this vector defines the total number of model parameters. If entries in this vector are named, the names are taken as parameter names. Names should be included if you'd like returned parameter vectors to have named entries, or if the functions `lambda`, `mu` or `rho` query parameter values by name (as opposed to numeric index).
`param_guess`	Numeric vector of size NP, specifying a first guess for the value of each model parameter. For fixed parameters, guess values are ignored. Can be `NULL` only if all model parameters are fixed.
`param_min`	Optional numeric vector of size NP, specifying lower bounds for model parameters. If of size 1, the same lower bound is applied to all parameters. Use `-Inf` to omit a lower bound for a parameter. If `NULL`, no lower bounds are applied. For fixed parameters, lower bounds are ignored.
`param_max`	Optional numeric vector of size NP, specifying upper bounds for model parameters. If of size 1, the same upper bound is applied to all parameters. Use `+Inf` to omit an upper bound for a parameter. If `NULL`, no upper bounds are applied. For fixed parameters, upper bounds are ignored.
`param_scale`	Optional numeric vector of size NP, specifying typical scales for model parameters. If of size 1, the same scale is assumed for all parameters. If `NULL`, scales are determined automatically. For fixed parameters, scales are ignored. It is strongly advised to provide reasonable scales, as this facilitates the numeric optimization algorithm.
`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to consider when calculating the likelihood. If this is equal to or greater than the root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.
`lambda`	Function specifying the speciation rate at any given age (time before present) and for any given parameter values. This function must take exactly two ar-

	guments, the 1st one being a numeric vector (one or more ages) and the 2nd one being a numeric vector of size NP (parameter values), and return a numeric vector of the same size as the 1st argument with strictly positive entries.
`mu`	Function specifying the extinction rate at any given age and for any given parameter values. This function must take exactly two arguments, the 1st one being a numeric vector (one or more ages) and the 2nd one being a numeric vector of size NP (parameter values), and return a numeric vector of the same size as the 1st argument with non-negative entries.
`rho`	Function specifying the sampling fraction (fraction of extant species sampled) for any given parameter values. This function must take exactly one argument, a numeric vector of size NP (parameter values), and return a numeric between 0 (exclusive) and 1 (inclusive).
`age_grid`	Numeric vector, specifying ages at which the `lambda` and `mu` functionals should be evaluated. This age grid must be fine enough to capture the possible variation in λ and μ over time, within the permissible parameter range. If of size 1, then lambda & mu are assumed to be time-independent. Listed ages must be strictly increasing, and must cover at least the full considered age interval (from 0 to `oldest_age`). Can also be `NULL` or a vector of size 1, in which case the speciation rate and extinction rate is assumed to be time-independent.
`condition`	Character, either "crown", "stem" or "none", specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age. "none" is usually not recommended.
`relative_dt`	Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time, when calculating the likelihood. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.
`Ntrials`	Integer, specifying the number of independent fitting trials to perform, each starting from a random choice of model parameters. Increasing `Ntrials` reduces the risk of reaching a non-global local maximum in the fitting objective.
`Nthreads`	Integer, specifying the number of parallel threads to use for performing multiple fitting trials simultaneously. This should generally not exceed the number of available CPUs on your machine. Parallel computing is not available on the Windows platform.
`max_model_runtime`	Optional numeric, specifying the maximum number of seconds to allow for each evaluation of the likelihood function. Use this to abort fitting trials leading to parameter regions where the likelihood takes a long time to evaluate (these are often unlikely parameter regions).
`fit_control`	Named list containing options for the `nlminb` optimization routine, such as `iter.max`, `eval.max` or `rel.tol`. For a complete list of options and default values see the documentation of `nlminb` in the `stats` package.

Details

This function is designed to estimate a finite set of scalar parameters ($p_1, .., p_n \in \mathbb{R}$) that determine the speciation rate λ, the extinction rate μ and the sampling fraction ρ, by maximizing the likelihood of observing a given timetree under the HBD model. For example, the investigator may assume that both λ and μ vary exponentially over time, i.e. they can be described by $\lambda(t) = \lambda_o \cdot e^{-\alpha t}$ and $\mu(t) = \mu_o \cdot e^{-\beta t}$ (where λ_o, μ_o are unknown present-day rates and α, β are unknown factors, and t is time before present), and that the sampling fraction ρ is known. In this case the model has 4 free parameters, $p_1 = \lambda_o$, $p_2 = \mu_o$, $p_3 = \alpha$ and $p_4 = \beta$, each of which may be fitted to the tree.

It is generally advised to provide as much information to the function `fit_hbd_model_parametric` as possible, including reasonable lower and upper bounds (`param_min` and `param_max`), a reasonable parameter guess (`param_guess`) and reasonable parameter scales `param_scale`. If some model parameters can vary over multiple orders of magnitude, it is advised to transform them so that they vary across fewer orders of magnitude (e.g., via log-transformation). It is also important that the `age_grid` is sufficiently fine to capture the variation of `lambda` and `mu` over time, since the likelihood is calculated under the assumption that both vary linearly between grid points.

Value

A list with the following elements:

`success`	Logical, indicating whether model fitting succeeded. If `FALSE`, the returned list will include an additional "error" element (character) providing a description of the error; in that case all other return variables may be undefined.
`objective_value`	The maximized fitting objective. Currently, only maximum-likelihood estimation is implemented, and hence this will always be the maximized log-likelihood.
`objective_name`	The name of the objective that was maximized during fitting. Currently, only maximum-likelihood estimation is implemented, and hence this will always be "loglikelihood".
`param_fitted`	Numeric vector of size NP (number of model parameters), listing all fitted or fixed model parameters in their standard order (see details above). If `param_names` was provided, elements in `fitted_params` will be named.
`param_guess`	Numeric vector of size NP, listing guessed or fixed values for all model parameters in their standard order. If `param_names` was provided, elements in `param_guess` will be named.
`loglikelihood`	The log-likelihood of the fitted model for the given timetree.
`NFP`	Integer, number of fitted (i.e., non-fixed) model parameters.
`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k - 2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`converged`	Logical, specifying whether the maximum likelihood was reached after convergence of the optimization algorithm. Note that in some cases the maximum likelihood may have been achieved by an optimization path that did not yet converge (in which case it's advisable to increase `iter.max` and/or `eval.max`).

`Niterations`	Integer, specifying the number of iterations performed during the optimization path that yielded the maximum likelihood.
`Nevaluations`	Integer, specifying the number of likelihood evaluations performed during the optimization path that yielded the maximum likelihood.

Author(s)

Stilianos Louca

References

H. Morlon, T. L. Parsons, J. B. Plotkin (2011). Reconciling molecular phylogenies with the fossil record. Proceedings of the National Academy of Sciences. 108:16327-16332.

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`simulate_deterministic_hbd`

`loglikelihood_hbd`

`fit_hbd_model_on_grid`

`fit_hbd_pdr_on_grid`

`fit_hbd_pdr_parametric`

Examples

```
## Not run:
# Generate a random tree with exponentially varying lambda & mu
Ntips     = 10000
rho       = 0.5 # sampling fraction
time_grid = seq(from=0, to=100, by=0.01)
lambdas   = 2*exp(0.1*time_grid)
mus       = 1.5*exp(0.09*time_grid)
tree      = generate_random_tree( parameters  = list(rarefaction=rho),
                                  max_tips    = Ntips/rho,
                                  coalescent  = TRUE,
                                  added_rates_times     = time_grid,
                                  added_birth_rates_pc  = lambdas,
                                  added_death_rates_pc  = mus)$tree
root_age = castor::get_tree_span(tree)$max_distance
cat(sprintf("Tree has %d tips, spans %g Myr\n",length(tree$tip.label),root_age))

# Define a parametric HBD model, with exponentially varying lambda & mu
# Assume that the sampling fraction is known
# The model thus has 4 parameters: lambda0, mu0, alpha, beta
lambda_function = function(ages,params){
return(params['lambda0']*exp(-params['alpha']*ages));
}
mu_function = function(ages,params){
```

```
return(params['mu0']*exp(-params['beta']*ages));
}
rho_function = function(params){
return(rho) # rho does not depend on any of the parameters
}

# Define an age grid on which lambda_function & mu_function shall be evaluated
# Should be sufficiently fine to capture the variation in lambda & mu
age_grid = seq(from=0,to=100,by=0.01)

# Perform fitting
# Lets suppose extinction rates are already known
cat(sprintf("Fitting model to tree..\n"))
fit = fit_hbd_model_parametric( tree,
                      param_values  = c(lambda0=NA, mu0=3, alpha=NA, beta=-0.09),
                      param_guess   = c(1,1,0,0),
                      param_min     = c(0,0,-1,-1),
                      param_max     = c(10,10,1,1),
                      param_scale   = 1, # all params are in the order of 1
                      lambda        = lambda_function,
                      mu            = mu_function,
                      rho           = rho_function,
                      age_grid      = age_grid,
                      Ntrials       = 10,    # perform 10 fitting trials
                      Nthreads      = 2,     # use 2 CPUs
                      max_model_runtime = 1, # limit model evaluation to 1 second
                      fit_control       = list(rel.tol=1e-6))
if(!fit$success){
cat(sprintf("ERROR: Fitting failed: %s\n",fit$error))
}else{
cat(sprintf("Fitting succeeded:\nLoglikelihood=%g\n",fit$loglikelihood))
print(fit)
}

## End(Not run)
```

`fit_hbd_pdr_on_grid` *Fit pulled diversification rates of birth-death models on a time grid.*

Description

Given an ultrametric timetree, estimate the pulled diversification rate of homogenous birth-death (HBD) models that best explains the tree via maximum likelihood. Every HBD model is defined by some speciation and extinction rates (λ and μ) over time, as well as the sampling fraction ρ (fraction of extant species sampled). “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates. For any given HBD model there exists an infinite number of alternative HBD models that predict the same deterministic lineages-through-time curve and yield the same likelihood for any given reconstructed timetree; these “congruent” models cannot be distinguished from one another solely based on the tree.

Each congruence class is uniquely described by the "pulled diversification rate" (PDR; Louca et al 2018), defined as $PDR = \lambda - \mu + \lambda^{-1} d\lambda/d\tau$ (where τ is time before present) as well as the product $\rho\lambda_o$ (where λ_o is the present-day speciation rate). That is, two HBD models are congruent if and only if they have the same PDR and the same product $\rho\lambda_o$. This function is designed to estimate the generating congruence class for the tree, by fitting the PDR on a grid of discrete times as well as the product $\rho\lambda_o$.

Usage

```
fit_hbd_pdr_on_grid(  tree,
                      oldest_age        = NULL,
                      age_grid          = NULL,
                      min_PDR           = -Inf,
                      max_PDR           = +Inf,
                      min_rholambda0    = 1e-10,
                      max_rholambda0    = +Inf,
                      guess_PDR         = NULL,
                      guess_rholambda0  = NULL,
                      fixed_PDR         = NULL,
                      fixed_rholambda0  = NULL,
                      splines_degree    = 1,
                      condition         = "stem",
                      relative_dt       = 1e-3,
                      Ntrials           = 1,
                      Nthreads          = 1,
                      max_model_runtime = NULL,
                      fit_control       = list())
```

Arguments

`tree`	An ultrametric timetree of class "phylo", representing the time-calibrated phylogeny of a set of extant species.
`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to consider when calculating the likelihood. If this is equal to or greater than the root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.
`age_grid`	Numeric vector, listing ages in ascending order at which the PDR is allowed to vary independently. This grid must cover at least the age range from 0 to `oldest_age`. If `NULL` or of length <=1 (regardless of value), then the PDR is assumed to be time-independent.
`min_PDR`	Numeric vector of length Ngrid (=`max(1,length(age_grid))`), or a single numeric, specifying lower bounds for the fitted PDR at each point in the age

	grid. If a single numeric, the same lower bound applies at all ages. Use `-Inf` to omit lower bounds.
`max_PDR`	Numeric vector of length Ngrid, or a single numeric, specifying upper bounds for the fitted PDR at each point in the age grid. If a single numeric, the same upper bound applies at all ages. Use `+Inf` to omit upper bounds.
`min_rholambda0`	Strictly positive numeric, specifying the lower bound for the fitted $\rho\lambda_o$ (sampling fraction times present-day extinction rate).
`max_rholambda0`	Strictly positive numeric, specifying the upper bound for the fitted $\rho\lambda_o$. Set to `+Inf` to omit this upper bound.
`guess_PDR`	Initial guess for the PDR at each age-grid point. Either `NULL` (an initial guess will be computed automatically), or a single numeric (guessing the same PDR at all ages) or a numeric vector of size Ngrid specifying a separate guess at each age-grid point. To omit an initial guess for some but not all age-grid points, set their guess values to `NA`. Guess values are ignored for non-fitted (i.e., fixed) parameters.
`guess_rholambda0`	Numeric, specifying an initial guess for the product $\rho\lambda_o$. If `NULL`, a guess will be computed automatically.
`fixed_PDR`	Optional fixed (i.e. non-fitted) PDR values on one or more age-grid points. Either `NULL` (PDR is not fixed anywhere), or a single numeric (PDR fixed to the same value at all grid points) or a numeric vector of size Ngrid (PDR fixed at one or more age-grid points, use `NA` for non-fixed values).
`fixed_rholambda0`	Numeric, optionally specifying a fixed value for the product $\rho\lambda_o$. If `NULL` or `NA`, the product $\rho\lambda_o$ is estimated.
`splines_degree`	Integer between 0 and 3 (inclusive), specifying the polynomial degree of the PDR between age-grid points. If 0, then the PDR is considered piecewise constant, if 1 then the PDR is considered piecewise linear, if 2 or 3 then the PDR is considered to be a spline of degree 2 or 3, respectively. The `splines_degree` influences the analytical properties of the curve, e.g. `splines_degree==1` guarantees a continuous curve, `splines_degree==2` guarantees a continuous curve and continuous derivative, and so on. A degree of 0 is generally not recommended.
`condition`	Character, either "crown" or "stem", specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age.
`relative_dt`	Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time, when calculating the likelihood. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

`Ntrials`	Integer, specifying the number of independent fitting trials to perform, each starting from a random choice of model parameters. Increasing `Ntrials` reduces the risk of reaching a non-global local maximum in the fitting objective.
`Nthreads`	Integer, specifying the number of parallel threads to use for performing multiple fitting trials simultaneously. This should generally not exceed the number of available CPUs on your machine. Parallel computing is not available on the Windows platform.
`max_model_runtime`	Optional numeric, specifying the maximum number of seconds to allow for each evaluation of the likelihood function. Use this to abort fitting trials leading to parameter regions where the likelihood takes a long time to evaluate (these are often unlikely parameter regions).
`fit_control`	Named list containing options for the `nlminb` optimization routine, such as `iter.max`, `eval.max` or `rel.tol`. For a complete list of options and default values see the documentation of `nlminb` in the `stats` package.

Details

It is generally advised to provide as much information to the function `fit_hbd_pdr_on_grid` as possible, including reasonable lower and upper bounds (`min_PDR`, `max_PDR`, `min_rholambda0` and `max_rholambda0`) and a reasonable parameter guess (`guess_PDR` and `guess_rholambda0`). It is also important that the `age_grid` is sufficiently fine to capture the expected major variations of the PDR over time, but keep in mind the serious risk of overfitting when `age_grid` is too fine and/or the tree is too small.

Value

A list with the following elements:

`success`	Logical, indicating whether model fitting succeeded. If `FALSE`, the returned list will include an additional “error” element (character) providing a description of the error; in that case all other return variables may be undefined.
`objective_value`	The maximized fitting objective. Currently, only maximum-likelihood estimation is implemented, and hence this will always be the maximized log-likelihood.
`objective_name`	The name of the objective that was maximized during fitting. Currently, only maximum-likelihood estimation is implemented, and hence this will always be “loglikelihood”.
`loglikelihood`	The log-likelihood of the fitted model for the given timetree.
`fitted_PDR`	Numeric vector of size Ngrid, listing fitted or fixed pulled diversification rates (PDR) at each age-grid point.
`fitted_rholambda0`	Numeric, specifying the fitted or fixed product $\rho\lambda 0$.
`guess_PDR`	Numeric vector of size Ngrid, specifying the initial guess for the PDR at each age-grid point.

`guess_rholambda0`	Numeric, specifying the initial guess for $\rho\lambda0$.
`age_grid`	The age-grid on which the PDR is defined. This will be the same as the provided `age_grid`, unless the latter was `NULL` or of length <=1.
`NFP`	Integer, number of fitted (i.e., non-fixed) parameters. If none of the PDRs or $\rho\lambda0$ were fixed, this will be equal to Ngrid+1.
`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k-2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`converged`	Logical, specifying whether the maximum likelihood was reached after convergence of the optimization algorithm. Note that in some cases the maximum likelihood may have been achieved by an optimization path that did not yet converge (in which case it's advisable to increase `iter.max` and/or `eval.max`).
`Niterations`	Integer, specifying the number of iterations performed during the optimization path that yielded the maximum likelihood.
`Nevaluations`	Integer, specifying the number of likelihood evaluations performed during the optimization path that yielded the maximum likelihood.

Author(s)

Stilianos Louca

References

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`simulate_deterministic_hbd`

`loglikelihood_hbd`

`fit_hbd_model_parametric`

`fit_hbd_model_on_grid`

`fit_hbd_pdr_parametric`

Examples

```
## Not run:
# Generate a random tree with exponentially varying lambda & mu
Ntips     = 10000
rho       = 0.5 # sampling fraction
time_grid = seq(from=0, to=100, by=0.01)
lambdas   = 2*exp(0.1*time_grid)
mus       = 1.5*exp(0.09*time_grid)
sim       = generate_random_tree( parameters  = list(rarefaction=rho),
                                  max_tips    = Ntips/rho,
                                  coalescent  = TRUE,
                                  added_rates_times     = time_grid,
                                  added_birth_rates_pc  = lambdas,
                                  added_death_rates_pc  = mus)
tree = sim$tree
root_age = castor::get_tree_span(tree)$max_distance
cat(sprintf("Tree has %d tips, spans %g Myr\n",length(tree$tip.label),root_age))

# calculate true PDR
lambda_slopes = diff(lambdas)/diff(time_grid);
lambda_slopes = c(lambda_slopes[1],lambda_slopes)
PDRs = lambdas - mus - (lambda_slopes/lambdas)


# Fit PDR on grid
Ngrid     = 10
age_grid  = seq(from=0,to=root_age,length.out=Ngrid)
fit = fit_hbd_pdr_on_grid(tree,
          age_grid    = age_grid,
          min_PDR     = -100,
          max_PDR     = +100,
          condition   = "crown",
          Ntrials     = 10,# perform 10 fitting trials
          Nthreads    = 10,# use two CPUs
          max_model_runtime = 1)  # limit model evaluation to 1 second
if(!fit$success){
  cat(sprintf("ERROR: Fitting failed: %s\n",fit$error))
}else{
  cat(sprintf("Fitting succeeded:\nLoglikelihood=%g\n",fit$loglikelihood))
  # plot fitted & true PDR
  plot( x     = fit$age_grid,
        y     = fit$fitted_PDR,
        main  = 'Fitted & true PDR',
        xlab  = 'age',
        ylab  = 'PDR',
        type  = 'b',
        col   = 'red',
        xlim  = c(root_age,0))
  lines(x     = sim$final_time-time_grid,
        y     = PDRs,
        type  = 'l',
        col   = 'blue');
}

## End(Not run)
```

fit_hbd_pdr_parametric

Fit parameterized pulled diversification rates of birth-death models.

Description

Given an ultrametric timetree, estimate the pulled diversification rate (PDR) of homogenous birth-death (HBD) models that best explains the tree via maximum likelihood, assuming that the PDR is given as a parameterized function of time before present. Every HBD model is defined by some speciation and extinction rates (λ and μ) over time, as well as the sampling fraction ρ (fraction of extant species sampled). “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates. For any given HBD model there exists an infinite number of alternative HBD models that predict the same deterministic lineages-through-time curve and yield the same likelihood for any given reconstructed timetree; these “congruent” models cannot be distinguished from one another solely based on the tree.

Each congruence class is uniquely described by its PDR, defined as $PDR = \lambda - \mu + \lambda^{-1} d\lambda/d\tau$ (where τ is time before present) as well as the product $\rho\lambda_o$ (where λ_o is the present-day speciation rate). That is, two HBD models are congruent if and only if they have the same PDR and the same product $\rho\lambda_o$. This function is designed to estimate the generating congruence class for the tree, by fitting a finite number of parameters defining the PDR and $\rho\lambda_o$.

Usage

```
fit_hbd_pdr_parametric(   tree,
  param_values,
  param_guess       = NULL,
  param_min         = -Inf,
  param_max         = +Inf,
  param_scale       = NULL,
  oldest_age        = NULL,
  PDR,
  rholambda0,
  age_grid          = NULL,
  condition         = "stem",
  relative_dt       = 1e-3,
  Ntrials           = 1,
  Nthreads          = 1,
  max_model_runtime = NULL,
  fit_control       = list())
```

Arguments

`tree`	An ultrametric timetree of class "phylo", representing the time-calibrated phylogeny of a set of extant species.
`param_values`	Numeric vector, specifying fixed values for a some or all model parameters. For fitted (i.e., non-fixed) parameters, use `NaN` or `NA`. For example, the vector `c(1.5,NA,40)` specifies that the 1st and 3rd model parameters are fixed at the values 1.5 and 40, respectively, while the 2nd parameter is to be fitted. The length of this vector defines the total number of model parameters. If entries in this vector are named, the names are taken as parameter names. Names should be included if you’d like returned parameter vectors to have named entries, or if the functions `PDR` or `rho` query parameter values by name (as opposed to numeric index).

`param_guess`	Numeric vector of size NP, specifying a first guess for the value of each model parameter. For fixed parameters, guess values are ignored. Can be `NULL` only if all model parameters are fixed.
`param_min`	Optional numeric vector of size NP, specifying lower bounds for model parameters. If of size 1, the same lower bound is applied to all parameters. Use `-Inf` to omit a lower bound for a parameter. If `NULL`, no lower bounds are applied. For fixed parameters, lower bounds are ignored.
`param_max`	Optional numeric vector of size NP, specifying upper bounds for model parameters. If of size 1, the same upper bound is applied to all parameters. Use `+Inf` to omit an upper bound for a parameter. If `NULL`, no upper bounds are applied. For fixed parameters, upper bounds are ignored.
`param_scale`	Optional numeric vector of size NP, specifying typical scales for model parameters. If of size 1, the same scale is assumed for all parameters. If `NULL`, scales are determined automatically. For fixed parameters, scales are ignored. It is strongly advised to provide reasonable scales, as this facilitates the numeric optimization algorithm.
`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to consider when calculating the likelihood. If this is equal to or greater than the root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.
`PDR`	Function specifying the pulled diversification rate at any given age (time before present) and for any given parameter values. This function must take exactly two arguments, the 1st one being a numeric vector (one or more ages) and the 2nd one being a numeric vector of size NP (parameter values), and return a numeric vector of the same size as the 1st argument.
`rholambda0`	Function specifying the product $\rho\lambda_o$ (sampling fraction times present-day speciation rate) for any given parameter values. This function must take exactly one argument, a numeric vector of size NP (parameter values), and return a strictly positive numeric.
`age_grid`	Numeric vector, specifying ages at which the `PDR` function should be evaluated. This age grid must be fine enough to capture the possible variation in the PDR over time, within the permissible parameter range. If of size 1, then the PDR is assumed to be time-independent. Listed ages must be strictly increasing, and must cover at least the full considered age interval (from 0 to `oldest_age`). Can also be `NULL` or a vector of size 1, in which case the PDR is assumed to be time-independent.
`condition`	Character, either "crown" or "stem", specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense

when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age.

`relative_dt` Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time, when calculating the likelihood. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

`Ntrials` Integer, specifying the number of independent fitting trials to perform, each starting from a random choice of model parameters. Increasing `Ntrials` reduces the risk of reaching a non-global local maximum in the fitting objective.

`Nthreads` Integer, specifying the number of parallel threads to use for performing multiple fitting trials simultaneously. This should generally not exceed the number of available CPUs on your machine. Parallel computing is not available on the Windows platform.

`max_model_runtime`
Optional numeric, specifying the maximum number of seconds to allow for each evaluation of the likelihood function. Use this to abort fitting trials leading to parameter regions where the likelihood takes a long time to evaluate (these are often unlikely parameter regions).

`fit_control` Named list containing options for the `nlminb` optimization routine, such as `iter.max`, `eval.max` or `rel.tol`. For a complete list of options and default values see the documentation of `nlminb` in the `stats` package.

Details

This function is designed to estimate a finite set of scalar parameters ($p_1, .., p_n \in \mathsf{R}$) that determine the PDR and the product $\rho\lambda_o$ (sampling fraction times present-dat extinction rate), by maximizing the likelihood of observing a given timetree under the HBD model. For example, the investigator may assume that the PDR varies exponentially over time, i.e. can be described by $PDR(t) = A \cdot e^{-Bt}$ (where A and B are unknown coefficients and t is time before present), and that the product $\rho\lambda_o$ is unknown. In this case the model has 3 free parameters, $p_1 = A$, $p_2 = B$ and $p_3 = \rho\lambda_o$, each of which may be fitted to the tree.

It is generally advised to provide as much information to the function `fit_hbd_pdr_parametric` as possible, including reasonable lower and upper bounds (`param_min` and `param_max`), a reasonable parameter guess (`param_guess`) and reasonable parameter scales `param_scale`. If some model parameters can vary over multiple orders of magnitude, it is advised to transform them so that they vary across fewer orders of magnitude (e.g., via log-transformation). It is also important that the `age_grid` is sufficiently fine to capture the variation of the PDR over time, since the likelihood is calculated under the assumption that both vary linearly between grid points.

Value

A list with the following elements:

`success` Logical, indicating whether model fitting succeeded. If `FALSE`, the returned list will include an additional "error" element (character) providing a description of the error; in that case all other return variables may be undefined.

`objective_value`	The maximized fitting objective. Currently, only maximum-likelihood estimation is implemented, and hence this will always be the maximized log-likelihood.
`objective_name`	The name of the objective that was maximized during fitting. Currently, only maximum-likelihood estimation is implemented, and hence this will always be "loglikelihood".
`param_fitted`	Numeric vector of size NP (number of model parameters), listing all fitted or fixed model parameters in their standard order (see details above). If `param_names` was provided, elements in `fitted_params` will be named.
`param_guess`	Numeric vector of size NP, listing guessed or fixed values for all model parameters in their standard order. If `param_names` was provided, elements in `param_guess` will be named.
`loglikelihood`	The log-likelihood of the fitted model for the given timetree.
`NFP`	Integer, number of fitted (i.e., non-fixed) model parameters.
`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k - 2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`converged`	Logical, specifying whether the maximum likelihood was reached after convergence of the optimization algorithm. Note that in some cases the maximum likelihood may have been achieved by an optimization path that did not yet converge (in which case it's advisable to increase `iter.max` and/or `eval.max`).
`Niterations`	Integer, specifying the number of iterations performed during the optimization path that yielded the maximum likelihood.
`Nevaluations`	Integer, specifying the number of likelihood evaluations performed during the optimization path that yielded the maximum likelihood.

Author(s)

Stilianos Louca

References

H. Morlon, T. L. Parsons, J. B. Plotkin (2011). Reconciling molecular phylogenies with the fossil record. Proceedings of the National Academy of Sciences. 108:16327-16332.

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`simulate_deterministic_hbd`

`loglikelihood_hbd`

`fit_hbd_model_on_grid`

`fit_hbd_model_parametric`

`fit_hbd_pdr_on_grid`

Examples

```
## Not run:
# Generate a random tree with exponentially varying lambda & mu
Ntips      = 10000
rho        = 0.5 # sampling fraction
time_grid = seq(from=0, to=100, by=0.01)
lambdas    = 2*exp(0.1*time_grid)
mus        = 1.5*exp(0.09*time_grid)
tree       = generate_random_tree( parameters  = list(rarefaction=rho),
                                   max_tips    = Ntips/rho,
                                   coalescent  = TRUE,
                                   added_rates_times     = time_grid,
                                   added_birth_rates_pc  = lambdas,
                                   added_death_rates_pc  = mus)$tree
root_age = castor::get_tree_span(tree)$max_distance
cat(sprintf("Tree has %d tips, spans %g Myr\n",length(tree$tip.label),root_age))

# Define a parametric HBD congruence class, with exponentially varying PDR
# The model thus has 3 parameters
PDR_function = function(ages,params){
return(params['A']*exp(-params['B']*ages));
}
rholambda0_function = function(params){
return(params['rholambda0'])
}

# Define an age grid on which lambda_function & mu_function shall be evaluated
# Should be sufficiently fine to capture the variation in the PDR
age_grid = seq(from=0,to=100,by=0.01)

# Perform fitting
# Lets suppose extinction rates are already known
cat(sprintf("Fitting class to tree..\n"))
fit = fit_hbd_pdr_parametric( tree,
                        param_values  = c(A=NA, B=NA, rholambda0=NA),
                        param_guess   = c(1,0,1),
                        param_min     = c(-10,-10,0),
                        param_max     = c(10,10,10),
                        param_scale   = 1, # all params are in the order of 1
                        PDR           = PDR_function,
                        rholambda0    = rholambda0_function,
                        age_grid      = age_grid,
                        Ntrials       = 10,    # perform 10 fitting trials
                        Nthreads      = 2,     # use 2 CPUs
                        max_model_runtime = 1, # limit model evaluation to 1 second
                        fit_control       = list(rel.tol=1e-6))
if(!fit$success){
cat(sprintf("ERROR: Fitting failed: %s\n",fit$error))
}else{
cat(sprintf("Fitting succeeded:\nLoglikelihood=%g\n",fit$loglikelihood))
print(fit)
}
```

```
## End(Not run)
```

`fit_hbd_psr_on_grid` *Fit pulled speciation rates of birth-death models on a time grid.*

Description

Given an ultrametric timetree, estimate the pulled speciation rate of homogenous birth-death (HBD) models that best explains the tree via maximum likelihood. Every HBD model is defined by some speciation and extinction rates (λ and μ) over time, as well as the sampling fraction ρ (fraction of extant species sampled). “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates. For any given HBD model there exists an infinite number of alternative HBD models that predict the same deterministic lineages-through-time curve and yield the same likelihood for any given reconstructed timetree; these “congruent” models cannot be distinguished from one another solely based on the tree.

Each congruence class is uniquely described by the “pulled speciation rate” (PSR), defined as the relative slope of the deterministic LTT over time, $PSR = -M^{-1}dM/d\tau$ (where τ is time before present). In other words, two HBD models are congruent if and only if they have the same PSR. This function is designed to estimate the generating congruence class for the tree, by fitting the PSR on a discrete time grid.

Usage

```
fit_hbd_psr_on_grid(  tree,
                      oldest_age        = NULL,
                      age_grid          = NULL,
                      min_PSR           = 0,
                      max_PSR           = +Inf,
                      guess_PSR         = NULL,
                      fixed_PSR         = NULL,
                      splines_degree    = 1,
                      condition         = "stem",
                      relative_dt       = 1e-3,
                      Ntrials           = 1,
                      Nthreads          = 1,
                      max_model_runtime = NULL,
                      fit_control       = list())
```

Arguments

`tree`	An ultrametric timetree of class "phylo", representing the time-calibrated phylogeny of a set of extant species.
`oldest_age`	Strictly positive numeric, specifying the oldest time before present (“age”) to consider when calculating the likelihood. If this is equal to or greater than the

root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.

`age_grid` Numeric vector, listing ages in ascending order at which the PSR is allowed to vary independently. This grid must cover at least the age range from 0 to `oldest_age`. If `NULL` or of length <=1 (regardless of value), then the PSR is assumed to be time-independent.

`min_PSR` Numeric vector of length Ngrid (=`max(1,length(age_grid))`), or a single numeric, specifying lower bounds for the fitted PSR at each point in the age grid. If a single numeric, the same lower bound applies at all ages. Note that the PSR is never negative.

`max_PSR` Numeric vector of length Ngrid, or a single numeric, specifying upper bounds for the fitted PSR at each point in the age grid. If a single numeric, the same upper bound applies at all ages. Use `+Inf` to omit upper bounds.

`guess_PSR` Initial guess for the PSR at each age-grid point. Either `NULL` (an initial guess will be computed automatically), or a single numeric (guessing the same PSR at all ages) or a numeric vector of size Ngrid specifying a separate guess at each age-grid point. To omit an initial guess for some but not all age-grid points, set their guess values to `NA`. Guess values are ignored for non-fitted (i.e., fixed) parameters.

`fixed_PSR` Optional fixed (i.e. non-fitted) PSR values on one or more age-grid points. Either `NULL` (PSR is not fixed anywhere), or a single numeric (PSR fixed to the same value at all grid points) or a numeric vector of size Ngrid (PSR fixed at one or more age-grid points, use `NA` for non-fixed values).

`splines_degree`
Integer between 0 and 3 (inclusive), specifying the polynomial degree of the PSR between age-grid points. If 0, then the PSR is considered piecewise constant, if 1 then the PSR is considered piecewise linear, if 2 or 3 then the PSR is considered to be a spline of degree 2 or 3, respectively. The `splines_degree` influences the analytical properties of the curve, e.g. `splines_degree==1` guarantees a continuous curve, `splines_degree==2` guarantees a continuous curve and continuous derivative, and so on. A degree of 0 is generally not recommended.

`condition` Character, either "crown" or "stem", specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age.

`relative_dt` Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time, when calculating the likelihood. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

`Ntrials`	Integer, specifying the number of independent fitting trials to perform, each starting from a random choice of model parameters. Increasing `Ntrials` reduces the risk of reaching a non-global local maximum in the fitting objective.
`Nthreads`	Integer, specifying the number of parallel threads to use for performing multiple fitting trials simultaneously. This should generally not exceed the number of available CPUs on your machine. Parallel computing is not available on the Windows platform.
`max_model_runtime`	Optional numeric, specifying the maximum number of seconds to allow for each evaluation of the likelihood function. Use this to abort fitting trials leading to parameter regions where the likelihood takes a long time to evaluate (these are often unlikely parameter regions).
`fit_control`	Named list containing options for the `nlminb` optimization routine, such as `iter.max`, `eval.max` or `rel.tol`. For a complete list of options and default values see the documentation of `nlminb` in the `stats` package.

Details

It is generally advised to provide as much information to the function `fit_hbd_psr_on_grid` as possible, including reasonable lower and upper bounds (`min_PSR` and `max_PSR`) and a reasonable parameter guess (`guess_PSR`). It is also important that the `age_grid` is sufficiently fine to capture the expected major variations of the PSR over time, but keep in mind the serious risk of overfitting when `age_grid` is too fine and/or the tree is too small.

Value

A list with the following elements:

`success`	Logical, indicating whether model fitting succeeded. If `FALSE`, the returned list will include an additional "error" element (character) providing a description of the error; in that case all other return variables may be undefined.
`objective_value`	The maximized fitting objective. Currently, only maximum-likelihood estimation is implemented, and hence this will always be the maximized log-likelihood.
`objective_name`	The name of the objective that was maximized during fitting. Currently, only maximum-likelihood estimation is implemented, and hence this will always be "loglikelihood".
`loglikelihood`	The log-likelihood of the fitted model for the given timetree.
`fitted_PSR`	Numeric vector of size Ngrid, listing fitted or fixed pulled speciation rates (PSR) at each age-grid point.
`guess_PSR`	Numeric vector of size Ngrid, specifying the initial guess for the PSR at each age-grid point.

`age_grid`	The age-grid on which the PSR is defined. This will be the same as the provided `age_grid`, unless the latter was `NULL` or of length <=1.
`NFP`	Integer, number of fitted (i.e., non-fixed) parameters. If none of the PSRs were fixed, this will be equal to Ngrid.
`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k - 2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`converged`	Logical, specifying whether the maximum likelihood was reached after convergence of the optimization algorithm. Note that in some cases the maximum likelihood may have been achieved by an optimization path that did not yet converge (in which case it's advisable to increase `iter.max` and/or `eval.max`).
`Niterations`	Integer, specifying the number of iterations performed during the optimization path that yielded the maximum likelihood.
`Nevaluations`	Integer, specifying the number of likelihood evaluations performed during the optimization path that yielded the maximum likelihood.

Author(s)

Stilianos Louca

References

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`simulate_deterministic_hbd`

`loglikelihood_hbd`

`fit_hbd_model_parametric`

`fit_hbd_model_on_grid`

`fit_hbd_pdr_parametric`

`fit_hbd_pdr_on_grid`

Examples

```
## Not run:
# Generate a random tree with exponentially varying lambda & mu
Ntips     = 10000
rho       = 0.5 # sampling fraction
time_grid = seq(from=0, to=100, by=0.01)
lambdas   = 2*exp(0.1*time_grid)
mus       = 1.5*exp(0.09*time_grid)
sim       = generate_random_tree( parameters  = list(rarefaction=rho),
                                  max_tips    = Ntips/rho,
                                  coalescent  = TRUE,
                                  added_rates_times     = time_grid,
                                  added_birth_rates_pc  = lambdas,
                                  added_death_rates_pc  = mus)
tree = sim$tree
root_age = castor::get_tree_span(tree)$max_distance
cat(sprintf("Tree has %d tips, spans %g Myr\n",length(tree$tip.label),root_age))

# Fit PSR on grid
oldest_age=root_age/2 # only consider recent times when fitting
Ngrid     = 10
age_grid  = seq(from=0,to=oldest_age,length.out=Ngrid)
fit = fit_hbd_psr_on_grid(tree,
          oldest_age  = oldest_age,
          age_grid    = age_grid,
          min_PSR     = 0,
          max_PSR     = +100,
          condition   = "crown",
          Ntrials     = 10,# perform 10 fitting trials
          Nthreads    = 10,# use two CPUs
          max_model_runtime = 1)  # limit model evaluation to 1 second
if(!fit$success){
  cat(sprintf("ERROR: Fitting failed: %s\n",fit$error))
}else{
  cat(sprintf("Fitting succeeded:\nLoglikelihood=%g\n",fit$loglikelihood))
  # plot fitted PSR
  plot( x     = fit$age_grid,
        y     = fit$fitted_PSR,
        main  = 'Fitted PSR',
        xlab  = 'age',
        ylab  = 'PSR',
        type  = 'b',
        xlim  = c(root_age,0))

 # plot deterministic LTT of fitted model
  plot( x     = fit$age_grid,
        y     = fit$fitted_LTT,
        main  = 'Fitted dLTT',
        xlab  = 'age',
        ylab  = 'lineages',
        type  = 'b',
        log   = 'y',
        xlim  = c(root_age,0))
}

## End(Not run)
```

`fit_musse` *Fit a discrete-state-dependent diversification model via maximum-likelihood.*

Description

The Binary State Speciation and Extinction (BiSSE) model (Maddison et al. 2007) and its extension to Multiple State Speciation Extinction (MuSSE) models (FitzJohn et al. 2009, 2012), Hidden State

Speciation Extinction (HiSSE) models (Beaulieu and O'meara, 2016) or Several Examined and Concealed States-dependent Speciation and Extinction (SecSSE) models (van Els et al. 2018), describe a Poissonian cladogenic process whose birth/death (speciation/extinction) rates depend on the states of an evolving discrete trait. Specifically, extant tips either go extinct or split continuously in time at Poissonian rates, and birth/death rates at each extant tip depend on the current state of the tip; lineages tansition stochastically between states acccording to a continuous-time Markov process with fixed transition rates.

This function takes as main input an ultrametric tree and a list of tip proxy states, and fits the parameters of a BiSSE/MuSSE/HiSSE/SecSSE model to the data via maximum-likelihood. Tips can have missing (unknown) proxy states, and the function can account for biases in species sampling and biases in the identification of proxy states. The likelihood is calculated using a mathematically equivalent, but computationally more efficient variant, of the classical postorder-traversal BiSSE/MuSSE/HiSSE/SecSSE algorithm, as described by Louca (2019). This function has been optimized for large phylogenetic trees, with a relatively small number of states (i.e. Nstates«Ntips); its time complexity scales roughly linearly with Ntips.

Usage

```
fit_musse(tree,
          Nstates,
          NPstates              = NULL,
          proxy_map             = NULL,
          state_names           = NULL,
          tip_pstates           = NULL,
          tip_priors            = NULL,
          sampling_fractions    = 1,
          reveal_fractions      = 1,
          transition_rate_model = "ARD",
          birth_rate_model      = "ARD",
          death_rate_model      = "ARD",
          transition_matrix     = NULL,
          birth_rates           = NULL,
          death_rates           = NULL,
          first_guess           = NULL,
          lower                 = NULL,
          upper                 = NULL,
          root_prior            = "likelihoods",
          root_conditioning     = "none",
          Ntrials               = 1,
          optim_algorithm       = "subplex",
          optim_max_iterations  = 10000,
          optim_max_evaluations = NULL,
          optim_rel_tol         = 1e-6,
          check_input           = TRUE,
          include_ancestral_likelihoods = FALSE,
          Nthreads              = 1,
          Nbootstraps           = 0,
          Ntrials_per_bootstrap = NULL,
          max_condition_number  = 1e4,
          relative_ODE_step     = 0.1,
          E_value_step          = 1e-4,
          D_temporal_resolution = 100,
          max_model_runtime     = NULL,
          verbose               = TRUE,
          verbose_prefix        = "")
```

Arguments

`tree`	Ultrametric phylogenetic tree of class "phylo", representing all or a subset of extant species.
`Nstates`	Integer, specifying the number of possible discrete states a tip can have, influencing speciation/extinction rates. For example, if `Nstates==2` then this corresponds to the common Binary State Speciation and Extinction (BiSSE) model (Maddison et al., 2007). In the case of a HiSSE/SecSSE model, `Nstates` refers to the total number of diversification rate categories. For example, in the case of the HiSSE model described by Beaulieu and O'meara (2016), `Nstates=4`.
`NPstates`	Integer, optionally specifying a number of "proxy-states" that are observed instead of the underlying speciation/extinction-modulating states. To fit a HiSSE/SecSSE model, `NPstates` should be smaller than `Nstates`. Each state corresponds to a different proxy-state, as defined using the variable `proxy_map` (see below). For BiSSE/MuSSE with no hidden states, `NPstates` can be set to either `NULL` or equal to `Nstates`; in either case, `NPstates` will be considered equal to `Nstates`. For example, in the case of the HiSSE model described by Beaulieu and O'meara (2016), `NPstates=2`.
`proxy_map`	Integer vector of size `Nstates` and with values in 1,..`NPstates`, specifying the correspondence between states (i.e. diversification-rate categories) and proxy-states, in a HiSSE/SecSSE model. Specifically, `proxy_map[s]` indicates which proxy-state the state s is represented by. Each proxy-state can represent multiple states (i.e. proxies are ambiguous), but each state must be represented by exactly one proxy-state. For example, to setup the HiSSE model described by Beaulieu and O'meara (2016), use `proxy_map=c(1,2,1,2)`. For non-HiSSE models, set this to `NULL` or to `c(1:Nstates)`. See below for more details.
`state_names`	Optional character vector of size `Nstates`, specifying a name/description for each state. This does not influence any of the calculations. It is merely used to add human-readable row/column names (rather than integers) to the returned vectors/matrices. If `NULL`, no row/column names are added.
`tip_pstates`	Integer vector of size Ntips, listing the proxy state at each tip, in the same order as tips are indexed in the tree. The vector may (but need not) include names; if it does, these are checked for consistency with the tree (if `check_input==TRUE`). Values must range from 1 to `NPstates` (which is assumed equal to `Nstates` in the case of BiSSE/MuSSE). States may also be `NA`, corresponding to unknown tip proxy states (no information available).
`tip_priors`	Numeric matrix of size Ntips x `Nstates` (or of size Ntips x `NPstates`), listing prior likelihoods of each state (or each proxy-state) at each tip. Can be

provided as an alternative to `tip_pstates`. Thus, `tip_priors[i,s]` is the likelihood of observing the data if the tip `i` was at state `s` (or proxy-state `s`). Either `tip_pstates` or `tip_priors` must be non-NULL, but not both.

`sampling_fractions`
: Integer vector of size `NPstates`, with values between 0 and 1, listing the sampling fractions of species depending on proxy-state. That is, `sampling_fractions[p]` is the probability that an extant species, having proxy state `p`, is included in the phylogeny. If all species are included in the tree with the same probability (i.e., independent of state), this can also be a single number. If `NULL` (default), all species are assumed to be included in the tree.

`reveal_fractions`
: Integer vector of size `NPstates`, with values between 0 and 1, listing the probabilities of proxy-state identification depending on proxy-state. That is, `reveal_fractions[p]` is the probability that a species with proxy-state `p` will have a known ("revealed") state, conditional upon being included in the tree. This can be used to incorporate reveal biases for tips, depending on their proxy state. Can also be `NULL` or a single number (in which case reveal fractions are assumed to be independent of proxy-state). Note that only the relative values in `reveal_fractions` matter, for example `c(1,2,1)` has the same effect as `c(0.5,1,0.5)`, because `reveal_fractions` is normalized internally anyway.

`transition_rate_model`
: Either a character or a 2D integer matrix of size Nstates x Nstates, specifying the model for the transition rates between states. This option controls the parametric complexity of the state transition model, i.e. the number of independent rates and the correspondence between independent and dependent rates. If a character, then it must be one of "ER", "SYM", "ARD", "SUEDE" or "SRD", as used for Mk models (see the function `asr_mk_model` for details). For example, "ARD" (all rates different) specifies that all transition rates should be considered as independent parameters with potentially different values.

  If an integer matrix, then it defines a custom parametric structure for the transition rates, by mapping entries of the transition matrix to a set of independent transition-rate parameters (numbered 1,2, and so on), similarly to the option `rate_model` in the function `asr_mk_model`, and as returned for example by the function `get_transition_index_matrix`. Entries must be between 1 and Nstates, however 0 may also be used to denote a fixed value of zero. For example, if `transition_rate_model[1,2]=transition_rate_model[2,1]`, then the transition rates 1->2 and 2->1 are assumed to be equal. Entries on the diagonal are ignored, since the diagonal elements are always adjusted to ensure a valid Markov transition matrix. To construct a custom matrix with the proper structure, it may be convenient to first generate an "ARD" matrix using `get_transition_index_matrix`, and then modify individual entries to reduce the number of independent rates.

`birth_rate_model`
: Either a character or an integer vector of length Nstates, specifying the model for the various birth (speciation) rates. This option controls the parametric complexity of the possible birth rates, i.e. the number of independent birth rates and the correspondence between independent and dependent birth rates. If a characteb,

then it must be either "ER" (equal rates) or "ARD" (all rates different). If an integer vector, it must map each state to an indepedent birth-rate parameter (indexed 1,2,..). For example, the vector `c(1,2,1)` specifies that the birth-rates λ_1 and λ_3 must be the same, but λ_2 is independent.

`death_rate_model`
: Either a character or an integer vector of length Nstates, specifying the model for the various death (extinction) rates. Similar to `birth_rate_model`.

`transition_matrix`
: Either `NULL` or a 2D matrix of size Nstates x Nstates, specifying known (and thus fixed) transition rates between states. For example, setting some elements to 0 specifies that these transitions cannot occur directly. May also contain `NA`, indicating rates that are to be fitted. If `NULL` or empty, all rates are considered unknown and are therefore fitted. Note that, unless `transition_rate_model=="ARD"`, values in `transition_matrix` are assumed to be consistent with the rate model, that is, rates specified to be equal under the transition rate model are expected to also have equal values in `transition_matrix`.

`birth_rates`
: Either `NULL`, or a single number, or a numeric vector of length Nstates, specifying known (and thus fixed) birth rates for each state. May contain `NA`, indicating rates that are to be fitted. For example, the vector `c(5,0,NA)` specifies that $\lambda_1 = 5$, $\lambda_2 = 0$ and that λ_3 is to be fitted. If `NULL` or empty, all birth rates are considered unknown and are therefore fitted. If a single number, all birth rates are considered fixed at that given value.

`death_rates`
: Either `NULL`, or a single number, or a numeric vector of length Nstates, specifying known (and thus fixed) death rates for each state. Similar to `birth_rates`.

`first_guess`
: Either `NULL`, or a named list containing optional initial suggestions for various model parameters, i.e. start values for fitting. The list can contain any or all of the following elements:

  - `transition_matrix`: A single number or a 2D numeric matrix of size Nstates x Nstates, specifying suggested start values for the transition rates. May contain NA, indicating rates that should be guessed automatically by the function. If a single number, then that value is used as a start value for all transition rates.
  - `birth_rates`: A single number or a numeric vector of size Nstates, specifying suggested start values for the birth rates. May contain NA, indicating rates that should be guessed automatically by the function (by fitting a simple birth-death model, see `fit_tree_model`).
  - `death_rates`: A single number or a numeric vector of size Nstates, specifying suggested start values for the death rates. May contain NA, indicating rates that should be guessed automatically by the function (by fitting a simple birth-death model, see `fit_tree_model`).

  Start values are only relevant for fitted (i.e., non-fixed) parameters.

`lower`
: Either `NULL` or a named list containing optional lower bounds for various model parameters. The list can contain any or all of the elements `transition_matrix`, `birth_rates` and `death_rates`, structured similarly to `first_guess`. For example, `list(transition_matrix=0.1,birth_rates=c(5,NA,NA))` specifies that all transition rates between states must be 0.1 or greater, that the

birth rate λ_1 must be 5 or greater, and that all other model parameters have unspecified lower bound. For parameters with unspecified lower bounds, zero is used as a lower bound. Lower bounds only apply to fitted (i.e., non-fixed) parameters.

`upper` Either `NULL` or a named list containing optional upper bounds for various model parameters. The list can contain any or all of the elements `transition_matrix`, `birth_rates` and `death_rates`, structured similarly to `upper`. For example, `list(transition_matrix=2,birth_rates=c(10,NA,NA))` specifies that all transition rates between states must be 2 or less, that the birth rate λ_1 must be 10 or less, and that all other model parameters have unspecified upper bound. For parameters with unspecified upper bounds, infinity is used as an upper bound. Upper bounds only apply to fitted (i.e., non-fixed) parameters.

`root_prior` Either a character or a numeric vector of size Nstates, specifying the prior probabilities of states for the root, i.e. the weights for obtaining a single model likelihood by averaging the root's state likelihoods. If a character, then it must be one of "flat", "empirical" or "likelihoods". "empirical" means the root's prior is set to the proportions of (estimated) extant species in each state (correcting for sampling fractions and reveal fractions, if applicable). "likelihoods" means that the computed state-likelihoods of the root are used, after normalizing to obtain a probability distribution; this is the approach used in the package `hisse::hisse` v1.8.9 under the option `root.p=NULL`, and the approach in the package `diversitree::find.mle` v0.9-10 under the option `root=ROOT.OBS`. If a numeric vector, `root_prior` specifies custom probabilities (weights) for each state.

`root_conditioning`
Character, specifying an optional modification to be applied to the root's state likelihoods prior to averaging. Can be "none" (no modification), "madfitz" or "herr_als". "madfitz" and "herr_als" (after van Els, Etiene and Herrera-Alsina 2018) are the options implemented in the package `hisse` v1.8.9, conditioning the root's state-likelihoods based on the birth-rates and the computed extinction probability (after or before averaging, respectively). See van Els (2018) for a comparison between "madfitz" and "herr_als".

`Ntrials` Non-negative integer, specifying the number of trials for fitting the model, using alternative (randomized) starting parameters at each trial. A larger `Ntrials` reduces the risk of landing on a local non-global optimum of the likelihood function, and thus increases the chances of finding the truly best fit. If 0, then no fitting is performed, and only the first-guess (i.e., provided or guessed start params) is evaluated and returned. Hence, setting `Ntrials=0` can be used to obtain a reasonable set of start parameters for subsequent fitting or for Markov Chain Monte Carlo.

`optim_algorithm`
Character, specifying the optimization algorithm for fitting. Must be one of either "optim", "nlminb" or "subplex".

`optim_max_iterations`
Integer, maximum number of iterations allowed for fitting. Only relevant for "optim" and "nlminb".

`optim_max_evaluations`
: Integer, maximum number of function evaluations allowed for fitting. Only relevant for "nlminb" and "subplex".

`optim_rel_tol`
: Numeric, relative tolerance for the fitted log-likelihood.

`check_input`
: Logical, specifying whether to check the validity of input variables. If you are certain that all input variables are valid, you can set this to `FALSE` to reduce computation.

`include_ancestral_likelihoods`
: Logical, specifying whether to include the state likelihoods for each node, in the returned variables. These are the "D" variables calculated as part of the likelihood based on the subtree descending from each node, and may be used for "local" ancestral state reconstructions.

`Nthreads`
: Integer, specifying the number of threads for running multiple fitting trials in parallel. Only relevant if `Ntrials>1`. Should generally not exceed the number of CPU cores on a machine. Must be a least 1.

`Nbootstraps`
: Integer, specifying an optional number of bootstrap samplings to perform, for estimating standard errors and confidence intervals of maximum-likelihood fitted parameters. If 0, no bootstrapping is performed. Typical values are 10-100. At each bootstrap sampling, a simulation of the fitted MuSSE/HiSSE model is performed, the parameters are estimated anew based on the simulation, and subsequently compared to the original fitted parameters. Each bootstrap sampling will thus use roughly as many computational resources as the original maximum-likelihood fit (e.g., same number of trials, same optimization parameters etc).

`Ntrials_per_bootstrap`
: Integer, specifying the number of fitting trials to perform for each bootstrap sampling. If `NULL`, this is set equal to `max(1,Ntrials)`. Decreasing `Ntrials_per_bootstrap` will reduce computation time, at the expense of potentially inflating the estimated confidence intervals; in some cases (e.g., for very large trees) this may be useful if fitting takes a long time and confidence intervals are very narrow anyway. Only relevant if `Nbootstraps>0`.

`max_condition_number`
: Positive unitless number, specifying the maximum permissible condition number for the "G" matrix computed for the log-likelihood. A higher condition number leads to faster computation (roughly on a log-scale) especially for large trees, at the potential expense of lower accuracy. Typical values are 1e2-1e5. See Louca (2019) for further details on the condition number of the G matrix.

`relative_ODE_step`
: Positive unitless number, specifying the default relative time step for the ordinary differential equation solvers.

`E_value_step`
: Positive unitless number, specifying the relative difference between subsequent recorded and interpolated E-values, in the ODE solver for the extinction probabilities E (Louca 2019). Typical values are 1e-2 to 1e-5. A smaller `E_value_step` increases interpolation accuracy, but also increases memory requirements and adds runtime (scaling with the tree's age span, not Ntips).

`D_temporal_resolution`
: Positive unitless number, specifying the relative resolution for interpolating G-map over time (Louca 2019). This is relative to the typical time scales at which G-map varies. For example, a resolution of 10 means that within a typical time scale there will be 10 interpolation points. Typical values are 1-1000. A greater resolution increases interpolation accuracy, but also increases memory requirements and adds runtime (scaling with the tree's age span, not Ntips).

`max_model_runtime`
: Numeric, optional maximum number of seconds for evaluating the likelihood of a model, prior to cancelling the calculation and returning Inf. This may be useful if extreme model parameters (e.g., reached transiently during fitting) require excessive calculation time. Parameters for which the calculation of the likelihood exceed this threshold, will be considered invalid and thus avoided during fitting. For example, for trees with 1000 tips a time limit of 10 seconds may be reasonable. If 0, no time limit is imposed.

`verbose`
: Logical, specifying whether to print progress reports and warnings to the screen. In any case, fatal errors are always reported.

`verbose_prefix`
: Character, specifying the line prefix for printing progress reports, warnings and errors to the screen.

Details

HiSSE/SecSSE models include two discrete traits, one trait that defines the rate categories of diversification rates (as in BiSSE/MuSSE), and one trait that does not itself influence diversification but whose states (here called "proxy states") each represent one or more of the diversity-modulating states. HiSSE models (Beaulieu and O'meara, 2016) and SecSSE models (van Els et al., 2018) are closely related to BiSSE/MuSSE models, the main difference being the fact that the actual diversification-modulating states are not directly observed. In essence, a HiSSE/SecSSE model is a BiSSE/MuSSE model, where the final tip states are replaced by their proxy states, thus "masking" the underlying diversity-modulating trait. This function is able to fit HiSSE/SecSSE models with appropriate choice of the input variables `Nstates`, `NPstates` and `proxy_map`. Note that the terminology and setup of HiSSE/SecSSE models followed here differs from their description in the original papers by Beaulieu and O'meara (2016) and van Els et al. (2018), in order to achieve what we think is a more intuitive unification of BiSSE/MuSSE/HiSSE/SecSSE. For ease of terminology, when considering a BiSSE/MuSSE model, here we use the terms "states" and "proxy-states" interchangeably, since under BiSSE/MuSSE the proxy trait can be considered identical to the diversification-modulating trait. A distinction between "states" and "proxy-states" is only relevant for HiSSE/SecSSE models.

As an example of a HiSSE model, `Nstates=4`, `NPstates=2` and `proxy_map=c(1,2,1,2)` specifies that states 1 and 3 are represented by proxy-state 1, and states 2 and 4 are represented by proxy-state 2. This is the original case described by Beaulieu and O'Meara (2016); in their terminology, there would be 2 "hidden"" states ("0" and "1") and 2 "observed" states ("A" and "B"), and the 4 diversification rate categories (`Nstates=4`) would be called "0A", "1A", "0B" and "1B". The somewhat different terminology used here allows for easier generalization to an arbitrary number of diversification-modulating states and an arbitrary number of proxy states. For example, if there are 6 diversification modulating states, represented by 3 proxy-states as 1->A, 2->A, 3->B, 4->C, 5->C, 6->C, then one would set `Nstates=6`, `NPstates=3` and `proxy_map=c(1,1,2,3,3,3)`.

The run time of this function scales asymptotically linearly with tree size (Ntips), although run times can vary substantially depending on model parameters. As a rule of thumb, the higher the birth/death/transition rates are compared to the tree's overall time span, the slower the calculation becomes.

The following arguments control the tradeoff between accuracy and computational efficiency:

- `max_condition_number`: A smaller value means greater accuracy, at longer runtime and more memory.
- `relative_ODE_step`: A smaller value means greater accuracy, at longer runtime.
- `E_value_step`: A smaller value means greater accuracy, at longer runtime and more memory.
- `D_temporal_resolution`: A greater value means greater accuracy, at longer runtime and more memory.

Typically, the default values for these arguments should be fine. For smaller trees, where cladogenic and sampling stochasticity is the main source of uncertainty, these parameters can probably be made less stringent (i.e., leading to lower accuracy and faster computation), but then again for small trees computational efficiency may not be an issue anyway.

Value

A named list with the following elements:

`success` Logical, indicating whether the fitting was successful. If `FALSE`, an additional element `error` (of type character) is included containing an explanation of the error; in that case the value of any of the other elements is undetermined.

`Nstates` Integer, the number of states assumed for the model.

`NPstates` Integer, the number of proxy states assumed for the model. Note that in the case of a BiSSE/MuSSE model, this will be the same as `Nstates`.

`parameters` Named list containing the final maximum-likelihood fitted model parameters. If `Ntrials>1`, then this contains the fitted parameters yielding the highest likelihood. Will contain the following elements:

- `transition_matrix`: 2D numeric matrix of size Nstates x Nstates, listing the fitted transition rates between states.
- `birth_rates`: Numeric vector of length Nstates, listing the fitted state-dependent birth rates.
- `death_rates`: Numeric vector of length Nstates, listing the fitted state-dependent death rates.

`start_parameters`
Named list containing the default start parameter values for the fitting. Structured similarly to `parameters`. Note that if `Ntrials>1`, only the first trial will have used these start values, all other trials will have used randomized start values. Will be defined even if `Ntrials==0`, and can thus be used to obtain a reasonable guess for the start parameters without actually fitting the model.

`loglikelihood`
The maximized log-likelihood of the model, if fitting succeeded.

`AIC`	The Akaike Information Criterion for the fitted model, defined as $2k - 2\log(L)$, where k is the number of fitted parameters and L is the maximized likelihood.
`Niterations`	The number of iterations needed for the best fit. Only relevant if the optimization method was "optim" or "nlminb".
`Nevaluations`	The number of function evaluations needed for the best fit. Only relevant if the optimization method was "nlminb" or "subplex".
`converged`	Logical, indicating whether convergence was successful during fitting. If convergence was not achieved, and the fitting was stopped due to one of the stopping criteria `optim_max_iterations` or `optim_max_evaluations`, the final likelihood will still be returned, but the fitted parameters may not be reasonable.
`warnings`	Character vector, listing any warnings encountered during evaluation of the likelihood function at the fitted parameter values. For example, this vector may contain warnings regarding the differential equation solvers or regarding the rank of the G-matrix (Louca, 2019).
`ML_root_state`	Integer between 1 and Nstates, an estimate of the root's state based on the computed state likelihoods.
`standard_errors`	Named list containing the elements "transition_matrix" (numeric matrix of size Nstates x Nstates), "birth_rates" (numeric vector of size Nstates) and "death_rates" (numeric vector of size Nstates), listing standard errors of all model parameters estimated using parametric bootstrapping. Only included if `Nbootstraps>0`. Note that the standard errors of non-fitted (i.e., fixed) parameters will be zero.
`CI50lower`	Named list containing the elements "transition_matrix" (numeric matrix of size Nstates x Nstates), "birth_rates" (numeric vector of size Nstates) and "death_rates" (numeric vector of size Nstates), listing the lower end of the 50% confidence interval (i.e. the 25% quantile) for each model parameter, estimated using parametric bootstrapping. Only included if `Nbootstraps>0`.
`CI50upper`	Similar to `CI50lower`, but listing the upper end of the 50% confidence interval (i.e. the 75% quantile) for each model parameter. For example, the confidence interval for he birth-rate λ_1 will be between `CI50lower$birth_rates[1]` and `CI50upper$birth_rates[1]`. Only included if `Nbootstraps>0`.
`CI95lower`	Similar to `CI50lower`, but listing the lower end of the 95% confidence interval (i.e. the 2.5% quantile) for each model parameter. Only included if `Nbootstraps>0`.
`CI95upper`	Similar to `CI50upper`, but listing the upper end of the 95% confidence interval (i.e. the 97.5% quantile) for each model parameter. Only included if `Nbootstraps>0`.
`CI`	2D numeric matrix, listing maximum-likelihood estimates, standard errors and confidence intervals for all model parameters (one row per parameter, one column for ML-estimates, one column for standard errors, two columns per confidence interval). Standard errors and confidence intervals are as estimated using parametric bootstrapping. This matrix contains the same information as `parameters`, `standard_errors`, `CI50lower`, `CI50upper`, `CI95lower` and `CI95upper`, but in a more compact format. Only included if `Nbootstraps>0`.

`ancestral_likelihoods`
: 2D matrix of size Nnodes x Nstates, listing the computed state-likelihoods for each node in the tree. These may be used for "local" ancestral state reconstructions, based on the information contained in the subtree descending from each node. Note that for each node the ancestral likelihoods have been normalized for numerical reasons, however they should not be interpreted as actual probabilities. For each node n and state s, `ancestral_likelihoods[n,s]` is proportional to the likelihood of observing the descending subtree and associated tip proxy states, if node n was at state s. Only included if `include_ancestral_likelihoods==TRUE`

Author(s)

Stilianos Louca

References

W. P. Maddison, P. E. Midford, S. P. Otto (2007). Estimating a binary character's effect on speciation and extinction. Systematic Biology. 56:701-710.

R. G. FitzJohn, W. P. Maddison, S. P. Otto (2009). Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Systematic Biology. 58:595-611

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

J. M. Beaulieu and B. C. O'Meara (2016). Detecting hidden diversification shifts in models of trait-dependent speciation and extinction. Systematic Biology. 65:583-601.

P. van Els, R. S. Etiene, L. Herrera-Alsina (2018). Detecting the dependence of diversification on multiple traits from phylogenetic trees and trait data. Systematic Biology. syy057.

S. L. Louca (2019). Modeling state-dependent diversification on large phylogenies. In review.

See Also

`simulate_dsse`, `asr_mk_model`, `fit_tree_model`

Examples

```
# EXAMPLE 1: BiSSE model
# - - - - - - - - - - - - - - -
# Choose random BiSSE model parameters
Nstates = 2
Q = get_random_mk_transition_matrix(Nstates, rate_model="ARD", max_rate=0.1)
parameters = list(birth_rates       = runif(Nstates,5,10),
                  death_rates       = runif(Nstates,0,5),
                  transition_matrix = Q)
rarefaction = 0.5 # randomly omit half of the tips

# Simulate a tree under the BiSSE model
simulation = simulate_musse(Nstates,
                            parameters         = parameters,
                            max_tips           = 1000,
                            sampling_fractions = rarefaction)
```

```
tree       = simulation$tree
tip_states = simulation$tip_states

## Not run:
# fit BiSSE model to tree & tip data
fit = fit_musse(tree,
                Nstates            = Nstates,
                tip_pstates        = tip_states,
                sampling_fractions = rarefaction)
if(!fit$success){
  cat(sprintf("ERROR: Fitting failed"))
}else{
  # compare fitted birth rates to true values
  errors = (fit$parameters$birth_rates - parameters$birth_rates)
  relative_errors = errors/parameters$birth_rates
  cat(sprintf("BiSSE relative birth-rate errors:\n"))
  print(relative_errors)
}

## End(Not run)


# EXAMPLE 2: HiSSE model, with bootstrapping
# - - - - - - - - - - - - - - -
# Choose random HiSSE model parameters
Nstates  = 4
NPstates = 2
Q = get_random_mk_transition_matrix(Nstates, rate_model="ARD", max_rate=0.1)
rarefaction = 0.5 # randomly omit half of the tips
parameters = list(birth_rates       = runif(Nstates,5,10),
                  death_rates       = runif(Nstates,0,5),
                  transition_matrix = Q)

# reveal the state of 30% & 60% of tips (in state 1 & 2, respectively)
reveal_fractions = c(0.3,0.6)

# use proxy map corresponding to Beaulieu and O'Meara (2016)
proxy_map = c(1,2,1,2)

# Simulate a tree under the HiSSE model
simulation = simulate_musse(Nstates,
                            NPstates            = NPstates,
                            proxy_map           = proxy_map,
                            parameters          = parameters,
                            max_tips            = 1000,
                            sampling_fractions  = rarefaction,
                            reveal_fractions    = reveal_fractions)
tree       = simulation$tree
tip_states = simulation$tip_proxy_states

## Not run:
# fit HiSSE model to tree & tip data
# run multiple trials to ensure global optimum
# also estimate confidence intervals via bootstrapping
fit = fit_musse(tree,
                Nstates            = Nstates,
                NPstates           = NPstates,
                proxy_map          = proxy_map,
                tip_pstates        = tip_states,
                sampling_fractions = rarefaction,
                reveal_fractions   = reveal_fractions,
                Ntrials            = 5,
                Nbootstraps        = 10)
if(!fit$success){
  cat(sprintf("ERROR: Fitting failed"))
}else{
  # compare fitted birth rates to true values
  errors = (fit$parameters$birth_rates - parameters$birth_rates)
  relative_errors = errors/parameters$birth_rates
  cat(sprintf("HiSSE relative birth-rate errors:\n"))
  print(relative_errors)

  # print 95%-confidence interval for first birth rate
  cat(sprintf("CI95 for lambda1: %g-%g",
              fit$CI95lower$birth_rates[1],
              fit$CI95upper$birth_rates[1]))
}

## End(Not run)
```

`fit_tree_model` *Fit a cladogenic model to an existing tree.*

Description

Fit the parameters of a tree generation model to an existing phylogenetic tree; branch lengths are assumed to be in time units. The fitted model is a stochastic cladogenic process in which speciations (births) and extinctions (deaths) are Poisson processes, as simulated by the function `generate_random_tree`. The birth and death rates of tips can each be constant or power-law functions of the number of extant tips. For example,

$$B = I + F \cdot N^E,$$

where B is the birth rate, I is the intercept, F is the power-law factor, N is the current number of extant tips and E is the power-law exponent. Each of the parameters I, F, E can be fixed or fitted.

Fitting can be performed via maximum-likelihood estimation, based on the waiting times between subsequent speciation and/or extinction events represented in the tree. Alternatively, fitting can be performed using least-squares estimation, based on the number of lineages represented in the tree over time ("diversity-vs-time" curve, a.k.a. "lineages-through-time"" curve). Note that the birth and death rates are NOT per-capita rates, they are absolute rates of species appearance and disappearance per time.

Usage

```
fit_tree_model( tree,
                parameters          = list(),
                first_guess         = list(),
                min_age             = 0,
                max_age             = 0,
                age_centile         = NULL,
                Ntrials             = 1,
                Nthreads            = 1,
                coalescent          = FALSE,
                discovery_fraction  = NULL,
                fit_control         = list(),
                min_R2              = -Inf,
                min_wR2             = -Inf,
                grid_size           = 100,
                max_model_runtime   = NULL,
                objective           = 'LL')
```

Arguments

`tree` A phylogenetic tree, in which branch lengths are assumed to be in time units. The tree may be a coalescent tree (i.e. only include extant clades) or a tree including extinct clades; the tree type influences what type of models can be fitted with each method.

`parameters` A named list specifying fixed and/or unknown birth-death model parameters, with one or more of the following elements:

- `birth_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which new species (tips) are added. In units 1/time.
- `birth_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which new species (tips) are added. In units 1/time.
- `birth_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which new species (tips) are added. Unitless.
- `death_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which extant species (tips) go extinct. In units 1/time.
- `death_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which extant species (tips) go extinct. In units 1/time.
- `death_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which extant species (tips) go extinct. Unitless.
- `resolution`: Numeric. Resolution at which the tree was collapsed (i.e. every node of age smaller than this resolution replaced by a single tip). In units time. A resolution of 0 means the tree was not collapsed.
- `rarefaction`: Numeric. Species sampling fraction, i.e. fraction of extant species represented (as tips) in the tree. A rarefaction of 1, for example, implies that the tree is complete, i.e. includes all extant species. Rarefaction is assumed to have occurred after collapsing.
- `extant_diversity`: The current total extant diversity, regardless of the rarefaction and resolution of the tree at hand. For example, if `resolution==0`

and `rarefaction==0.5` and the tree has 1000 tips, then `extant_diversity` should be `2000`. If `resolution` is fixed at 0 and `rarefaction` is also fixed, this can be left `NULL` and will be inferred automatically by the function.

Each of the above elements can also be `NULL`, in which case the parameter is fitted. Elements can also be vectors of size 2 (specifying constraint intervals), in which case the parameters are fitted and constrained within the intervals specified. For example, to fit `death_rate_factor` while constraining it to the interval [1,2], set its value to `c(1,2)`.

`first_guess`	A named list (with entries named as in `parameters`) specifying starting values for any of the fitted model parameters. Note that if `Ntrials>1`, then start values may be randomly modified in all but the first trial. For any parameters missing from `first_guess`, initial values are always randomly chosen. `first_guess` can also be `NULL`.
`min_age`	Numeric. Minimum distance from the tree crown, for a node/tip to be considered in the fitting. If <=0 or `NULL`, this constraint is ignored. Use this option to omit most recent nodes.
`max_age`	Numeric. Maximum distance from the tree crown, for a node/tip to be considered in the fitting. If <=0 or `NULL`, this constraint is ignored. Use this option to omit old nodes, e.g. with highly uncertain placements.
`age_centile`	Numeric within 0 and 1. Fraction of youngest nodes/tips to consider for the fitting. This can be used as an alternative to max_age. E.g. if set to 0.6, then the 60% youngest nodes/tips are considered. Either `age_centile` or `max_age` must be non-NULL, but not both.
`Ntrials`	Integer. Number of fitting attempts to perform, each time using randomly varied start values for fitted parameters. The returned fitted parameter values will be taken from the trial with greatest achieved fit objective. A larger number of trials will decrease the chance of hitting a local non-global optimum during fitting.
`Nthreads`	Number of threads to use for parallel execution of multiple fitting trials. On Windows, this option has no effect because Windows does not support forks.
`coalescent`	Logical, specifying whether the input tree is a coalescent tree (and thus the coalescent version of the model should be fitted). Only available if `objective=='R2'`.
`discovery_fraction`	Function handle, mapping age to the fraction of discovered lineages in a tree. That is, `discovery_fraction(tau)` is the probability that a lineage at age `tau`, that has an extant descendant today, will be represented (discovered) in the coalescent tree. In particular, `discovery_fraction(0)` equals the fraction of extant lineages represented in the tree. If this is provided, then `parameters$rarefaction` is fixed to 1, and `discovery_fraction` is applied after simulation. Only relevant if `coalescent==TRUE`. Experimental, so leave this `NULL` if you don't know what it means.
`fit_control`	Named list containing options for the `stats::nlminb` optimization routine, such as `eval.max` (max number of evaluations), `iter.max` (max number of iterations) and `rel.tol` (relative tolerance for convergence).
`min_R2`	Minimum coefficient of determination of the diversity curve (clade counts vs time) of the model when compared to the input tree, for a fitted model to be

	accepted. For example, if set to 0.5 then only fit trials achieving an R2 of at least 0.5 will be considered. Set this to `-Inf` to not filter fitted models based on the R2.
`min_wR2`	Similar to `min_R2`, but applying to the weighted R2, where squared-error weights are proportional to the inverse squared diversities.
`grid_size`	Integer. Number of equidistant time points to consider when calculating the R2 of a model's diversity-vs-time curve.
`max_model_runtime`	Numeric. Maximum runtime (in seconds) allowed for each model evaluation during fitting. Use this to escape from badly parameterized models during fitting (this will likely cause the affected fitting trial to fail). If `NULL` or <=0, this option is ignored.
`objective`	Character. Objective function to optimize during fitting. Can be either "LL" (log-likelihood of waiting times between speciation events and between extinction events), "R2" (coefficient of determination of diversity-vs-time curve), "wR2" (weighted R2, where weights of squared errors are proportional to the inverse diversities observed in the tree) or "lR2" (logarithmic R2, i.e. R2 calculated for the logarithm of the diversity-vs-time curve). Note that "wR2" will weight errors at lower diversities more strongly than "R2".

Value

A named list with the following elements:

`success`	Logical, indicating whether the fitting was successful.
`objective_value`	Numeric. The achieved maximum value of the objective function (log-likelihood, R2 or weighted R2).
`parameters`	A named list listing all model parameters (fixed and fitted).
`start_parameters`	A named list listing the start values of all model parameters. In the case of multiple fitting trials, this will list the initial (non-randomized) guess.
`R2`	Numeric. The achieved coefficient of determination of the fitted model, based on the diversity-vs-time curve.
`wR2`	Numeric. The achieved weighted coefficient of determination of the fitted model, based on the diversity-vs-time curve. Weights of squared errors are proportional to the inverse squared diversities observed in the tree.
`lR2`	Numeric. The achieved coefficient of determination of the fitted model on a log axis, i.e. based on the logarithm of the diversity-vs-time curve.
`Nspeciations`	Integer. Number of speciation events (=nodes) considered during fitting. This only includes speciations visible in the tree.
`Nextinctions`	Integer. Number of extinction events (=non-crown tips) considered during fitting. This only includes extinctions visible in the tree, i.e. tips whose distance from the root is lower than the maximum.
`grid_times`	Numeric vector. Time points considered for the diversity-vs-time curve. Times will be constrained between `min_age` and `max_age` if these were specified.

`tree_diversities`
: Number of lineages represented in the tree through time, calculated for each of `grid_times`.

`model_diversities`
: Number of lineages through time as predicted by the model (in the deterministic limit), calculated for each of `grid_times`. If `coalescent==TRUE` then these are the number of lineages expected to be represented in the coalescent tree (this may be lower than the actual number of extant clades at any given time point, if the model includes extinctions).

`fitted_parameter_names`
: Character vector, listing the names of fitted (i.e. non-fixed) parameters.

`locally_fitted_parameters`
: Named list of numeric vectors, listing the fitted values for each parameter and for each fitting trial. For example, if `birth_rate_factor` was fitted, then `locally_fitted_parameters$birth_rate_factor` will be a numeric vector of size `Ntrials` (or less, if some trials failed or omitted), listing the locally-optimized values of the parameter for each considered fitting trial. Mainly useful for diagnostic purposes.

`objective` Character. The name of the objective function used for fitting ("LL", "R2" or "wR2").

`Ntips` The number of tips in the input tree.

`Nnodes` The number of nodes in the input tree.

`min_age` The minimum age of nodes/tips considered during fitting.

`max_age` The maximum age of nodes/tips considered during fitting.

`age_centile` Numeric or `NULL`, equal to the `age_centile` specified as input to the function.

Author(s)

Stilianos Louca

See Also

`generate_random_tree`, `simulate_diversification_model` `reconstruct_past_diversificatio`

Examples

```
# Generate a tree using a simple speciation model
parameters = list(birth_rate_intercept  = 1,
                  birth_rate_factor     = 0,
                  birth_rate_exponent   = 0,
                  death_rate_intercept  = 0,
                  death_rate_factor     = 0,
                  death_rate_exponent   = 0,
                  resolution            = 0,
                  rarefaction           = 1)
tree = generate_random_tree(parameters, max_tips=100)
```

```
# Fit model to the tree
fitting_parameters = parameters
fitting_parameters$birth_rate_intercept = NULL # fit only this parameter
fitting = fit_tree_model(tree,fitting_parameters)

# compare fitted to true value
T = parameters$birth_rate_intercept
F = fitting$parameters$birth_rate_intercept
cat(sprintf("birth_rate_intercept: true=%g, fitted=%g\n",T,F))
```

generate_random_tree

Generate a tree using a Poissonian speciation/extinction model.

Description

Generate a random timetree via simulation of a Poissonian speciation/extinction (birth/death) process. New species are added (born) by splitting of a randomly chosen extant tip. The tree-wide birth and death rates of tips can each be constant or power-law functions of the number of extant tips. For example,

$$B = I + F \cdot N^E,$$

where B is the tree-wide birth rate (species generation rate), I is the intercept, F is the power-law factor, N is the current number of extant tips and E is the power-law exponent. Optionally, the per-capita (tip-specific) birth and death rates can be extended by adding a custom time series provided by the user.

Usage

```
generate_random_tree(parameters            = list(),
                     max_tips              = NULL,
                     max_time              = NULL,
                     max_time_eq           = NULL,
                     coalescent            = TRUE,
                     as_generations        = FALSE,
                     Nsplits               = 2,
                     added_rates_times     = NULL,
                     added_birth_rates_pc = NULL,
                     added_death_rates_pc = NULL,
                     added_periodic        = FALSE,
                     tip_basename          = "",
                     node_basename         = NULL,
                     include_birth_times   = FALSE,
                     include_death_times   = FALSE)
```

Arguments

`parameters`	A named list specifying the birth-death model parameters, with one or more of the following entries: `birth_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which new species (tips) are added. In units 1/time. By default this is 0. `birth_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which new species (tips) are added. In units 1/time. By default this is 0. `birth_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which new species (tips) are added. Unitless. By default this is 1. `death_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which extant species (tips) go extinct. In units 1/time. By default this is 0. `death_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which extant species (tips) go extinct. In units 1/time. By default this is 0. `death_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which extant species (tips) go extinct. Unitless. By default this is 1. `resolution`: Non-negative numeric, specifying the resolution (in time units) at which to collapse the final tree by combining closely related tips. Any node whose age is smaller than this threshold, will be represented by a single tip. Set `resolution=0` to not collapse tips (default). `rarefaction`: Numeric between 0 and 1. Rarefaction to be applied to the final tree (fraction of random tips kept in the tree). Note that if `coalescent==FALSE`, rarefaction may remove both extant as well as extinct clades. Set `rarefaction=1` to not perform any rarefaction (default).
`max_tips`	Maximum number of tips of the tree to be generated. If `coalescent=TRUE`, this refers to the number of extant tips. Otherwise, it refers to the number of extinct + extant tips. If `NULL` or <=0, the number of tips is unlimited (so be careful).
`max_time`	Maximum duration of the simulation. If `NULL` or <=0, this constraint is ignored.
`max_time_eq`	Maximum duration of the simulation, counting from the first point at which speciation/extinction equilibrium is reached, i.e. when (birth rate - death rate) changed sign for the first time. If `NULL` or <0, this constraint is ignored.
`coalescent`	Logical, specifying whether only the coalescent tree (i.e. the tree spanning the extant tips) should be returned. If `coalescent==FALSE` and the death rate is non-zero, then the tree may include non-extant tips (i.e. tips whose distance from the root is less than the total time of evolution). In that case, the tree will not be ultrametric.
`as_generations`	Logical, specifying whether edge lengths should correspond to generations. If FALSE, then edge lengths correspond to time.

`Nsplits`	Integer greater than 1. Number of child-tips to generate at each diversification event. If set to 2, the generated tree will be bifurcating. If >2, the tree will be multifurcating.
`added_rates_times`	Numeric vector, listing time points (in ascending order) for the custom per-capita birth and/or death rates time series (see `added_birth_rates_pc` and `added_death_rates_pc` below). Can also be `NULL`, in which case the custom time series are ignored.
`added_birth_rates_pc`	Numeric vector of the same size as `added_rates_times`, listing per-capita birth rates to be added to the power law part. Can also be `NULL`, in which case this option is ignored and birth rates are purely described by the power law.
`added_death_rates_pc`	Numeric vector of the same size as `added_rates_times`, listing per-capita death rates to be added to the power law part. Can also be `NULL`, in which case this option is ignored and death rates are purely described by the power law.
`added_periodic`	Logical, indicating whether `added_birth_rates_pc` and `added_death_rates_pc` should be extended periodically if needed (i.e. if not defined for the entire simulation time). If `FALSE`, added birth & death rates are extended with zeros.
`tip_basename`	Character. Prefix to be used for tip labels (e.g. "tip."). If empty (""), then tip labels will be integers "1", "2" and so on.
`node_basename`	Character. Prefix to be used for node labels (e.g. "node."). If `NULL`, no node labels will be included in the tree.
`include_birth_times`	Logical. If `TRUE`, then the times of speciation events (in order of occurrence) will also be returned.
`include_death_times`	Logical. If `TRUE`, then the times of extinction events (in order of occurrence) will also be returned.

Details

If `max_time==NULL`, then the returned tree will always contain `max_tips` tips. In particular, if at any moment during the simulation the tree only includes a single extant tip, the death rate is temporarily set to zero to prevent the complete extinction of the tree. If `max_tips==NULL`, then the simulation is ran as long as specified by `max_time`. If neither `max_time` nor `max_tips` is `NULL`, then the simulation halts as soon as the time exceeds `max_time` or the number of tips (extant tips if `coalescent` is `TRUE`) exceeds `max_tips`. If `max_tips!=NULL` and `Nsplits>2`, then the last diversification even may generate fewer than `Nsplits` children, in order to keep the total number of tips within the specified limit.

If `rarefaction<1` and `resolution>0`, collapsing of closely related tips (at the resolution specified) takes place prior to rarefaction (i.e., subsampling applies to the already collapsed tips).

Both the per-capita birth and death rates can be made into completely arbitrary functions of time, by setting all power-law coefficients to zero and providing custom time series `added_birth_rates_pc` and `added_death_rates_pc`.

Value

A named list with the following elements:

`success`	Logical, indicating whether the tree was successfully generated. If `FALSE`, the only other value returned is `error`.
`tree`	A rooted bifurcating (if `Nsplits==2`) or multifurcating (if `Nsplits>2`) tree of class "phylo", generated according to the specified birth/death model. If `coalescent==TRUE` or if all death rates are zero, and only if `as_generations==FALSE`, then the tree will be ultrametric. If `as_generations==TRUE` and `coalescent==FALSE`, all edges will have unit length.
`root_time`	Numeric, giving the time at which the tree's root was first split during the simulation. Note that if `coalescent==TRUE`, this may be later than the first speciation event during the simulation.
`final_time`	Numeric, giving the final time at the end of the simulation. Note that if `coalescent==TRUE`, then this may be greater than the total time span of the tree (since the root of the coalescent tree need not correspond to the first speciation event).
`equilibrium_time`	Numeric, giving the first time where the sign of (death rate - birth rate) changed from the beginning of the simulation, i.e. when speciation/extinction equilibrium was reached. May be infinite if the simulation stoped before reaching this point.
`Nbirths`	Total number of birth events (speciations) that occurred during tree growth. This may be lower than the total number of tips in the tree if death rates were non-zero and `coalescent==TRUE`, or if `Nsplits>2`.
`Ndeaths`	Total number of deaths (extinctions) that occurred during tree growth.
`Ncollapsed`	Number of tips removed from the tree while collapsing at the resolution specified.
`Nrarefied`	Number of tips removed from the tree due to rarefaction.
`birth_times`	Numeric vector, listing the times of speciation events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`death_times`	Numeric vector, listing the times of extinction events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`error`	Character, containing an explanation of ther error that occurred. Only included if `success==FALSE`.

Author(s)

Stilianos Louca

References

D. J. Aldous (2001). Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. Statistical Science. 16:23-34.

M. Steel and A. McKenzie (2001). Properties of phylogenetic trees generated by Yule-type speciation models. Mathematical Biosciences. 170:91-112.

Examples

```
# Simple speciation model
parameters = list(birth_rate_intercept=1)
tree = generate_random_tree(parameters,max_tips=100)$tree

# Exponential growth rate model
parameters = list(birth_rate_factor=1)
tree = generate_random_tree(parameters,max_tips=100)$tree
```

---

generate_tree_with_evolving_rates

Generate a random tree with evolving speciation/extinction rates.

Description

Generate a random phylogenetic tree via simulation of a Poissonian speciation/extinction (birth/death) process. New species are added (born) by splitting of a randomly chosen extant tip. Per-capita birth and death rates (aka. speciation and extinction rates) evolve under some stochastic process (e.g. Brownian motion) along each edge. Thus, the probability rate of a tip splitting or going extinct depends on the tip, with closely related tips having more similar per-capita birth and death rates.

Usage

```
generate_tree_with_evolving_rates(parameters           = list(),
                                  rate_model           = 'BM',
                                  max_tips             = NULL,
                                  max_time             = NULL,
                                  max_time_eq          = NULL,
                                  coalescent           = TRUE,
                                  as_generations       = FALSE,
                                  Nsplits              = 2,
                                  tip_basename         = "",
                                  node_basename        = NULL,
                                  include_birth_times  = FALSE,
                                  include_death_times  = FALSE,
                                  include_rates        = FALSE)
```

Arguments

`parameters`	A named list specifying the model parameters for the evolving birth/death rates. The precise entries expected depend on the chosen `rate_model` (see details below).
`rate_model`	Character, specifying the model for the evolving per-capita birth/death rates. Must be one of the following: 'BM' (Brownian motion constrained to a finite interval via reflection), 'Mk' (discrete-state continuous-time Markov chain with fixed transition rates).

`max_tips`	Maximum number of tips of the tree to be generated. If `coalescent=TRUE`, this refers to the number of extant tips. Otherwise, it refers to the number of extinct + extant tips. If `NULL` or <=0, the number of tips is unlimited (so be careful).
`max_time`	Maximum duration of the simulation. If `NULL` or <=0, this constraint is ignored.
`max_time_eq`	Maximum duration of the simulation, counting from the first point at which speciation/extinction equilibrium is reached, i.e. when (birth rate - death rate) changed sign for the first time. If `NULL` or <0, this constraint is ignored.
`coalescent`	Logical, specifying whether only the coalescent tree (i.e. the tree spanning the extant tips) should be returned. If `coalescent==FALSE` and the death rate is non-zero, then the tree may include non-extant tips (i.e. tips whose distance from the root is less than the total time of evolution). In that case, the tree will not be ultrametric.
`as_generations`	Logical, specifying whether edge lengths should correspond to generations. If FALSE, then edge lengths correspond to time.
`Nsplits`	Integer greater than 1. Number of child-tips to generate at each diversification event. If set to 2, the generated tree will be bifurcating. If >2, the tree will be multifurcating.
`tip_basename`	Character. Prefix to be used for tip labels (e.g. "tip."). If empty (""), then tip labels will be integers "1", "2" and so on.
`node_basename`	Character. Prefix to be used for node labels (e.g. "node."). If `NULL`, no node labels will be included in the tree.
`include_birth_times`	Logical. If `TRUE`, then the times of speciation events (in order of occurrence) will also be returned.
`include_death_times`	Logical. If `TRUE`, then the times of extinction events (in order of occurrence) will also be returned.
`include_rates`	Logical. If `TRUE`, then the bper-capita birth & death rates of all tips and nodes will also be returned.

Details

If `max_time==NULL`, then the returned tree will always contain `max_tips` tips. In particular, if at any moment during the simulation the tree only includes a single extant tip, the death rate is temporarily set to zero to prevent the complete extinction of the tree. If `max_tips==NULL`, then the simulation is ran as long as specified by `max_time`. If neither `max_time` nor `max_tips` is `NULL`, then the simulation halts as soon as the time exceeds `max_time` or the number of tips (extant tips if `coalescent` is `TRUE`) exceeds `max_tips`. If `max_tips!=NULL` and `Nsplits>2`, then the last diversification even may generate fewer than `Nsplits` children, in order to keep the total number of tips within the specified limit.

If `rate_model=='BM'`, then per-capita birth rates (speciation rates) and per-capita death rates (extinction rates) evolve according to Brownian Motion, constrained to a finite interval via reflection. Note that speciation and extinction rates are only updated at branching points, i.e. during speciation events, while waiting times until speciation/extinction are based on rates at the previous branching point. The argument `parameters` should be a named list including one or more of the following elements:

- `birth_rate_diffusivity`: Non-negative number. Diffusivity constant for the Brownian motion model of the evolving per-capita birth rate. In units 1/time^3. See `simulate_bm_model` for an explanation of the diffusivity parameter.
- `min_birth_rate_pc`: Non-negative number. The minimum allowed per-capita birth rate of a clade. In units 1/time. By default this is 0.
- `max_birth_rate_pc`: Non-negative number. The maximum allowed per-capita birth rate of a clade. In units 1/time. By default this is 1.
- `death_rate_diffusivity`: Non-negative number. Diffusivity constant for the Brownian motion model of the evolving per-capita death rate. In units 1/time^3. See `simulate_bm_model` for an explanation of the diffusivity parameter.
- `min_death_rate_pc`: Non-negative number. The minimum allowed per-capita death rate of a clade. In units 1/time. By default this is 0.
- `max_death_rate_pc`: Non-negative number. The maximum allowed per-capita death rate of a clade. In units 1/time. By default this is 1.
- `root_birth_rate_pc`: Non-negative number, between `min_birth_rate_pc` and `max_birth_rate_pc`, specifying the initial per-capita birth rate of the root. If left unspecified, this will be chosen randomly and uniformly within the allowed interval.
- `root_death_rate_pc`: Non-negative number, between `min_death_rate_pc` and `max_death_rate_pc`, specifying the initial per-capita death rate of the root. If left unspecified, this will be chosen randomly and uniformly within the allowed interval.
- `rarefaction`: Numeric between 0 and 1. Rarefaction to be applied at the end of the simulation (fraction of random tips kept in the tree). Note that if `coalescent==FALSE`, rarefaction may remove both extant as well as extinct clades. Set `rarefaction=1` to not perform any rarefaction.

If `rate_model=='Mk'`, then speciation/extinction rates are determined by a tip's current "state", which evolves according to a continuous-time discrete-state Markov chain (Mk model) with constant transition rates. The argument `parameters` should be a named list including one or more of the following elements:

- `Nstates`: Number of possible discrete states a tip can have. For example, if `Nstates` then this corresponds to the common Binary State Speciation and Extinction (BiSSE) model (Maddison et al., 2007). By default this is 1.
- `state_birth_rates`: Numeric vector of size Nstates, listing the per-capita birth rate (speciation rate) at each state. Can also be a single number (all states have the same birth rate).
- `state_death_rates`: Numeric vector of size Nstates, listing the per-capita death rate (extinction rate) at each state. Can also be a single number (all states have the same death rate).
- `transition_matrix`: 2D numeric matrix of size Nstates x Nstates. Transition rate matrix for the Markov chain model of birth/death rate evolution.

- `start_state`: Integer within 1,..,`Nstates`, specifying the initial state of the first created lineage. If left unspecified, this is chosen randomly and uniformly among all possible states.
- `rarefaction`: Same as when `rate_model=='BM'`.

Note: The option `rate_model=='Mk'` is deprecated and included for backward compatibility purposes only. To generate a tree with Markov transitions between states (known as Multiple State Speciation and Extinction model), use the command `simulate_dsse` instead.

Value

A named list with the following elements:

`success`	Logical, indicating whether the simulation was successful. If `FALSE`, an additional element `error` (of type character) is included containing an explanation of the error; in that case the value of any of the other elements is undetermined.
`tree`	A rooted bifurcating (if `Nsplits==2`) or multifurcating (if `Nsplits>2`) tree of class "phylo", generated according to the specified birth/death model. If `coalescent==TRUE` or if all death rates are zero, and only if `as_generations==FALSE`, then the tree will be ultrametric. If `as_generations==TRUE` and `coalescent==FALSE`, all edges will have unit length.
`root_time`	Numeric, giving the time at which the tree's root was first split during the simulation. Note that if `coalescent==TRUE`, this may be later than the first speciation event during the simulation.
`final_time`	Numeric, giving the final time at the end of the simulation. If `coalescent==TRUE`, then this may be greater than the total time span of the tree (since the root of the coalescent tree need not correspond to the first speciation event).
`equilibrium_time`	Numeric, giving the first time where the sign of (death rate - birth rate) changed from the beginning of the simulation, i.e. when speciation/extinction equilibrium was reached. May be infinite if the simulation stoped before reaching this point.
`Nbirths`	Total number of birth events (speciations) that occurred during tree growth. This may be lower than the total number of tips in the tree if death rates were non-zero and `coalescent==TRUE`, or if `Nsplits>2`.
`Ndeaths`	Total number of deaths (extinctions) that occurred during tree growth.
`birth_times`	Numeric vector, listing the times of speciation events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`death_times`	Numeric vector, listing the times of extinction events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`birth_rates_pc`	Numeric vector, listing the per-capita birth rate of each tip and node in the tree. The length of an edge in the tree was thus drawn from an exponential distribution with rate equal to the per-capita birth rate of the child tip or node.
`death_rates_pc`	Numeric vector, listing the per-capita death rate of each tip and node in the tree.

`states`	Integer vector of size Ntips+Nnodes, listing the discrete state of each tip and node in the tree. Only included if `rate_model=="Mk"`.
`start_state`	Integer, specifying the initial state of the first created lineage (either provided during the function call, or generated randomly). Only included if `rate_model=="Mk"`.
`root_birth_rate_pc`	Numeric, specifying the initial per-capita birth rate of the root (either provided during the function call, or generated randomly). Only included if `rate_model=="BM"`.
`root_death_rate_pc`	Numeric, specifying the initial per-capita death rate of the root (either provided during the function call, or generated randomly). Only included if `rate_model=="BM"`.

Author(s)

Stilianos Louca

References

D. J. Aldous (2001). Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. Statistical Science. 16:23-34.

W. P. Maddison, P. E. Midford, S. P. Otto (2007). Estimating a binary character's effect on speciation and extinction. Systematic Biology. 56:701-710.

See Also

`simulate_dsse`

Examples

```
# Example 1
# Generate tree, with rates evolving under Brownian motion
parameters = list(birth_rate_diffusivity  = 1,
                  min_birth_rate_pc       = 1,
                  max_birth_rate_pc       = 2,
                  death_rate_diffusivity  = 0.5,
                  min_death_rate_pc       = 0,
                  max_death_rate_pc       = 1)
simulation = generate_tree_with_evolving_rates(parameters,
                                               rate_model='BM',
                                               max_tips=1000,
                                               include_rates=TRUE)
tree  = simulation$tree
Ntips = length(tree$tip.label)

# plot per-capita birth & death rates of tips
plot( x=simulation$birth_rates_pc[1:Ntips],
      y=simulation$death_rates_pc[1:Ntips],
      type='p',
      xlab="pc birth rate",
      ylab="pc death rate",
      main="Per-capita birth & death rates across tips (BM model)",
        las=1)


######################
# Example 2
# Generate tree, with rates evolving under a binary-state model
Q = get_random_mk_transition_matrix(Nstates=2, rate_model="ER", max_rate=0.1)
parameters = list(Nstates = 2,
                  state_birth_rates = c(1,1.5),
                  state_death_rates = 0.5,
                  transition_matrix = Q)
simulation = generate_tree_with_evolving_rates(parameters,
                                               rate_model='Mk',
                                               max_tips=1000,
                                               include_rates=TRUE)
tree  = simulation$tree
Ntips = length(tree$tip.label)

# plot distribution of per-capita birth rates of tips
rates = simulation$birth_rates_pc[1:Ntips]
barplot(table(rates)/length(rates),
        xlab="rate",
        main="Distribution of pc birth rates across tips (Mk model)")
```

get_all_distances_to_root

Get distances of all tips and nodes to the root.

Description

Given a rooted phylogenetic tree, calculate the phylogenetic distance (cumulative branch length) of the root to each tip and node.

Usage

```
get_all_distances_to_root(tree, as_edge_count=FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`as_edge_count`	Logical, specifying whether distances should be counted in number of edges, rather than cumulative edge length. This is the same as if all edges had length 1.

Details

If `tree$edge.length` is missing, then every edge in the tree is assumed to be of length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The asymptotic average time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

A numeric vector of size Ntips+Nnodes, with the i-th element being the distance (cumulative branch length) of the i-th tip or node to the root. Tips are indexed 1,..,Ntips and nodes are indexed (Ntips+1),..,(Ntips+Nnodes).

Author(s)

Stilianos Louca

See Also

`get_pairwise_distances`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1,
                              death_rate_intercept=0.5),
                            max_tips=Ntips)$tree

# calculate distances to root
all_distances = get_all_distances_to_root(tree)

# extract distances of nodes to root
node_distances = all_distances[(Ntips+1):(Ntips+tree$Nnode)]

# plot histogram of distances (across all nodes)
hist(node_distances, xlab="distance to root", ylab="# nodes", prob=FALSE);
```

`get_all_node_depths` *Get the phylogenetic depth of each node in a tree.*

Description

Given a rooted phylogenetic tree, calculate the phylogenetic depth of each node (mean distance to its descending tips).

Usage

```
get_all_node_depths(tree, as_edge_count=FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`as_edge_count`	Logical, specifying whether distances should be counted in number of edges, rather than cumulative edge length. This is the same as if all edges had length 1.

Details

If `tree$edge.length` is missing, then every edge in the tree is assumed to be of length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The asymptotic average time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

A numeric vector of size Nnodes, with the i-th element being the mean distance of the i-th node to all of its tips.

Author(s)

Stilianos Louca

See Also

`get_all_distances_to_root`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1,
                                 death_rate_intercept=0.5),
                            max_tips=Ntips)$tree

# calculate node phylogenetic depths
node_depths = get_all_node_depths(tree)

# plot histogram of node depths
hist(node_depths, xlab="phylogenetic depth", ylab="# nodes", prob=FALSE);
```

`get_all_pairwise_distances`

Get distances between all pairs of tips and/or nodes.

Description

Calculate phylogenetic ("patristic") distances between all pairs of tips or nodes in the tree, or among a subset of tips/nodes requested.

Usage

```
get_all_pairwise_distances( tree,
                            only_clades     = NULL,
                            as_edge_counts  = FALSE,
                            check_input     = TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`only_clades`	Optional integer vector or character vector, listing tips and/or nodes to which to restrict pairwise distance calculations. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes). If a character vector, it must list tip and/or node names. For example, if `only_clades=c('apple','lemon','pear')`, then only the distance between ‘apple' and ‘lemon', between ‘apple' and 'pear', and between ‘lemon' and ‘pear' are calculated. If `only_clades==NULL`, then this is equivalent to `only_clades=c(1:(Ntips+Nnodes))`.
`check_input`	Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.
`as_edge_counts`	Logical, specifying whether distances should be calculated in terms of edge counts, rather than cumulative edge lengths. This is the same as if all edges had length 1.

Details

The "patristic distance" between two tips and/or nodes is the shortest cumulative branch length that must be traversed along the tree in order to reach one tip/node from the other.This function returns a square distance matrix, containing the patristic distance between all possible pairs of tips/nodes in the tree (or among the ones provided in `only_clades`).

If `tree$edge.length` is missing, then each edge is assumed to be of length 1; this is the same as setting `as_edge_counts=TRUE`. The tree may include multi-furcations as well as monofurcations (i.e. nodes with only one child). The input tree must be rooted at some node for technical reasons (see function `root_at_node`), but the choice of the root node does not influence the result. If `only_clades` is a character vector, then `tree$tip.label` must exist. If node names are included in `only_clades`, then `tree$node.label` must also exist.

The asymptotic average time complexity of this function for a balanced binary tree is O(NC*NC*Nanc + Ntips), where NC is the number of tips/nodes considered (e.g., the length of `only_clades`) and Nanc is the average number of ancestors per tip.

Value

A 2D numeric matrix of size NC x NC, where NC is the number of tips/nodes considered, and with the entry in row r and column c listing the distance between the r-th and the c-th clade considered (e.g., between clades `only_clades[r]` and `only_clades[c]`). Note that if `only_clades` was specified, then the rows and columns in the returned distance matrix correspond to the entries in `only_clades` (i.e., in the same order). If `only_clades` was `NULL`, then the rows and columns in the returned distance matrix correspond to tips (1,..,Ntips) and nodes (Ntips+1,..,Ntips+Nnodes)

Author(s)

Stilianos Louca

See Also

`get_all_distances_to_root`, `get_pairwise_distances`

Examples

```
# generate a random tree
Ntips = 100
tree  = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# calculate distances between all internal nodes
only_clades = c((Ntips+1):(Ntips+tree$Nnode))
distances = get_all_pairwise_distances(tree, only_clades)

# reroot at some other node
tree = root_at_node(tree, new_root_node=20, update_indices=FALSE)
new_distances = get_all_pairwise_distances(tree, only_clades)

# verify that distances remained unchanged
plot(distances,new_distances,type='p')
```

`get_independent_contrasts`
Phylogenetic independent contrasts for continuous traits.

Description

Calculate phylogenetic independent contrasts (PICs) for one or more continuous traits on a phylogenetic tree, as described by Felsenstein (1985). The trait states are assumed to be known for all tips of the tree. PICs are commonly used to calculate correlations between multiple traits, while accounting for shared evolutionary history at the tips. This function also returns an estimate for the state of the root or, equivalently, the phylogenetically weighted mean of the tip states (Garland et al., 1999).

Usage

```
get_independent_contrasts(tree,
                          tip_states,
                          scaled = TRUE,
                          only_bifurcations = FALSE,
                          check_input = TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, or a 2D numeric matrix of size Ntips x Ntraits, specifying the numeric state of each trait at each tip in the tree.

`scaled`	Logical, specifying whether to divide (standardize) PICs by the square root of their expected variance, as recommended by Felsenstein (1985).
`only_bifurcations`	Logical, specifying whether to only calculate PICs for bifurcating nodes. If `FALSE`, then multifurcations are temporarily expanded to bifurcations, and an additional PIC is calculated for each created bifurcation. If `TRUE`, then multifurcations are not expanded and PICs will not be calculated for them.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

If the tree is bifurcating, then one PIC is returned for each node. If multifurcations are present and `only_bifurcations==FALSE`, these are internally expanded to bifurcations and an additional PIC is returned for each such bifurcation. PICs are never returned for monofurcating nodes. Hence, in general the number of returned PICs is the number of bifurcations in the tree, potentially after multifurcations have been expanded to bifurcations (if `only_bifurcations==FALSE`).

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multifurcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child). Edges with length 0 will be adjusted internally to some tiny length (chosen to be much smaller than the smallest non-zero length).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

The function has asymptotic time complexity O(Nedges x Ntraits). It is more efficient to calculate PICs of multiple traits with the same function call, than to calculate PICs for each trait separately. For a single trait, this function is equivalent to the function `ape::pic`, with the difference that it can handle multifurcating trees.

Value

A list with the following elements:

`PICs`	A numeric vector (if `tip_states` is a vector) or a numeric matrix (if `tip_states` is a matrix), listing the phylogenetic independent contrasts for each trait and for each bifurcating node (potentially after multifurcations have been expanded). If a matrix, then `PICs[:,T]` will list the PICs for the T-th trait. Note that the order of elements in this vector (or rows, if `PICs` is a matrix) is not necessarily the order of nodes in the tree, and that `PICs` may contain fewer or more elements (or rows) than there were nodes in the input tree.
`distances`	Numeric vector of the same size as `PICs`. The "evolutionary distances" (or time) corresponding to the PICs under a Brownian motion model of trait evolution. These roughly correspond to the cumulative edge lengths between sister nodes from which PICs were calculated; hence their units are the same as those of edge lengths. They do not take into account the actual trait values. See Felsenstein (1985) for details.

`nodes` Integer vector of the same size as `PICs`, listing the node indices for which PICs are returned. If `only_bifurcations==FALSE`, then this vector may contain `NA`s, corresponding to temporary nodes created during expansion of multifurcations.

If `only_bifurcations==TRUE`, then this vector will only list nodes that were bifurcating in the input tree. In that case, `PICs[1]` will correspond to the node with name `tree$node.label[nodes[1]]`, whereas `PICs[2]` will correspond to the node with name `tree$node.label[nodes[2]]`, and so on.

`root_state` Numeric vector of size Ntraits, listing the globally estimated state for the root or, equivalently, the phylogenetically weighted mean of the tip states.

`root_standard_error`
Numeric vector of size Ntraits, listing the phylogenetically estimated standard errors of the root state under a Brownian motion model. The standard errors have the same units as the traits and depend both on the tree topology as well as the tip states. Calculated according to the procedure described by Garland et al. (1999, page 377).

`root_CI95` Numeric vector of size Ntraits, listing the radius (half width) of the 95% confidence interval of the root state. Calculated according to the procedure described by Garland et al. (1999, page 377). Note that in contrast to the CI95 returned by the `ace` function in the `ape` package (v. 0.5-64), `root_CI95` has the same units as the traits and depends both on the tree topology as well as the tip states.

Author(s)

Stilianos Louca

References

J. Felsenstein (1985). Phylogenies and the Comparative Method. The American Naturalist. 125:1-15.

T. Garland Jr., P. E. Midford, A. R. Ives (1999). An introduction to phylogenetically based statistical methods, with a new method for confidence intervals on ancestral values. American Zoologist. 39:374-388.

See Also

`asr_independent_contrasts`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# simulate a continuous trait
tip_states = simulate_bm_model(tree, diffusivity=0.1, include_nodes=FALSE)$tip_states;

# calculate PICs
```

```
results = get_independent_contrasts(tree, tip_states, scaled=TRUE, only_bifurcations=TRUE)

# assign PICs to the bifurcating nodes in the input tree
PIC_per_node = rep(NA, tree$Nnode)
valids = which(!is.na(results$nodes))
PIC_per_node[results$nodes[valids]] = results$PICs[valids]
```

---

get_mrca_of_set *Most recent common ancestor of a set of tips/nodes.*

Description

Given a rooted phylogenetic tree and a set of tips and/or nodes ("descendants"), calculate the most recent common ancestor (MRCA) of those descendants.

Usage

```
get_mrca_of_set(tree, descendants)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`descendants`	An integer vector or character vector, specifying the tips/nodes for which to find the MRCA. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes), where Ntips and Nnodes is the number of tips and nodes in the tree, respectively. If a character vector, it must list tip and/or node names. In this case `tree` must include `tip.label`, as well as `node.label` if nodes are included in `descendants`.

Details

The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). Duplicate entries in `descendants` are ignored.

Value

An integer in 1,..,(Ntips+Nnodes), representing the MRCA using the same index as in `tree$edge`. If the MRCA is a tip, then this index will be in 1,..,Ntips. If the MRCA is a node, then this index will be in (Ntips+1),..,(Ntips+Nnodes).

Author(s)

Stilianos Louca

See Also

`get_pairwise_mrcas`, `get_tips_for_mrcas`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# pick 3 random tips or nodes
descendants = sample.int(n=(Ntips+tree$Nnode), size=3, replace=FALSE)

# calculate MRCA of picked descendants
mrca = get_mrca_of_set(tree, descendants)
```

---

get_pairwise_distances

Get distances between pairs of tips or nodes.

Description

Calculate phylogenetic ("patristic") distances between tips or nodes in some list A and tips or nodes in a second list B of the same size.

Usage

```
get_pairwise_distances(tree, A, B, as_edge_counts=FALSE, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`A`	An integer vector or character vector of size Npairs, specifying the first of the two members of each pair for which to calculate the distance. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes). If a character vector, it must list tip and/or node names.
`B`	An integer vector or character vector of size Npairs, specifying the second of the two members of each pair for which to calculate the distance. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes). If a character vector, it must list tip and/or node names.
`check_input`	Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.
`as_edge_counts`	Logical, specifying whether distances should be calculated in terms of edge counts, rather than cumulative edge lengths. This is the same as if all edges had length 1.

Details

The "patristic distance" between two tips and/or nodes is the shortest cumulative branch length that must be traversed along the tree in order to reach one tip/node from the other. Given a list of tips and/or nodes A, and a 2nd list of tips and/or nodes B of the same size, this function will calculate patristic distance between each pair (A[i], B[i]), where i=1,2,..,Npairs.

If `tree$edge.length` is missing, then each edge is assumed to be of length 1; this is the same as setting `as_edge_counts=TRUE`. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The input tree must be rooted at some node for technical reasons (see function `root_at_node`), but the choice of the root node does not influence the result. If `A` and/or `B` is a character vector, then `tree$tip.label` must exist. If node names are included in `A` and/or `B`, then `tree$node.label` must also exist.

The asymptotic average time complexity of this function for a balanced binary tree is O(Ntips+Npairs*log2(Ntips)).

Value

A numeric vector of size Npairs, with the i-th element being the patristic distance between the tips/nodes A[i] and B[i].

Author(s)

Stilianos Louca

See Also

`get_all_distances_to_root`, `get_all_pairwise_distances`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# pick 3 random pairs of tips or nodes
Npairs = 3
A = sample.int(n=(Ntips+tree$Nnode), size=Npairs, replace=FALSE)
B = sample.int(n=(Ntips+tree$Nnode), size=Npairs, replace=FALSE)

# calculate distances
distances = get_pairwise_distances(tree, A, B)

# reroot at some other node
tree = root_at_node(tree, new_root_node=20, update_indices=FALSE)
new_distances = get_pairwise_distances(tree, A, B)

# verify that distances remained unchanged
print(distances)
print(new_distances)
```

`get_pairwise_mrcas` *Get most recent common ancestors of tip/node pairs.*

Description

Given a rooted phylogenetic tree and one or more pairs of tips and/or nodes, for each pair of tips/nodes find the most recent common ancestor (MRCA). If one clade is descendant of the other clade, the latter will be returned as MRCA.

Usage

```
get_pairwise_mrcas(tree, A, B, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`A`	An integer vector or character vector of size Npairs, specifying the first of the two members of each pair of tips/nodes for which to find the MRCA. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes). If a character vector, it must list tip and/or node names.
`B`	An integer vector or character vector of size Npairs, specifying the second of the two members of each pair of tips/nodes for which to find the MRCA. If an integer vector, it must list indices of tips (from 1 to Ntips) and/or nodes (from Ntips+1 to Ntips+Nnodes). If a character vector, it must list tip and/or node names.
`check_input`	Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.

Details

The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). If `tree$edge.length` is missing, then each edge is assumed to be of length 1. Note that in some cases the MRCA of two tips may be a tip, namely when both tips are the same.

If `A` and/or `B` is a character vector, then `tree$tip.label` must exist. If node names are included in `A` and/or `B`, then `tree$node.label` must also exist.

The asymptotic average time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

An integer vector of size Npairs with values in 1,..,Ntips (tips) and/or in (Ntips+1),..,(Ntips+Nnodes) (nodes), with the i-th element being the index of the MRCA of tips/nodes `A[i]` and `B[i]`.

Author(s)

Stilianos Louca

See Also

`get_mrca_of_set`, `get_tips_for_mrcas`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# pick 3 random pairs of tips or nodes
Npairs = 3
A = sample.int(n=(Ntips+tree$Nnode), size=Npairs, replace=FALSE)
B = sample.int(n=(Ntips+tree$Nnode), size=Npairs, replace=FALSE)

# calculate MRCAs
MRCAs = get_pairwise_mrcas(tree, A, B)
```

---

get_random_diffusivity_matrix

Create a random diffusivity matrix for a Brownian motion model.

Description

Create a random diffusivity matrix for a Brownian motion model of multi-trait evolution. This may be useful for testing purposes. The diffusivity matrix is drawn from the Wishart distribution of symmetric, nonnegative-definite matrixes:

$$D = X^T \cdot X, \quad X[i,j] \sim N(0,V), \quad i = 1,..,n, j = 1,..,p,$$

where n is the degrees of freedom, p is the number of traits and V is a scalar scaling.

Usage

```
get_random_diffusivity_matrix(Ntraits, degrees=NULL, V=1)
```

Arguments

`Ntraits`	The number of traits modelled. Equal to the number of rows and the number of columns of the returned matrix.
`degrees`	Degrees of freedom for the Wishart distribution. Must be equal to or greater than `Ntraits`. Can also be `NULL`, which is the same as setting it equal to `Ntraits`.
`V`	Positive number. A scalar scaling for the Wishart distribution.

Value

A real-valued quadratic symmetric non-negative definite matrix of size Ntraits x Ntraits. Almost surely (in the probabilistic sense), this matrix will be positive definite.

Author(s)

Stilianos Louca

See Also

`get_random_mk_transition_matrix`, `simulate_bm_model`

Examples

```
# generate a 5x5 diffusivity matrix
D = get_random_diffusivity_matrix(Ntraits=5)

# check that it is indeed positive definite
if(all(eigen(D)$values>0)){
  cat("Indeed positive definite\n");
}else{
  cat("Not positive definite\n");
}
```

`get_random_mk_transition_matrix`
Create a random transition matrix for an Mk model.

Description

Create a random transition matrix for a fixed-rates continuous-time Markov model of discrete trait evolution ("Mk model"). This may be useful for testing purposes.

Usage

```
get_random_mk_transition_matrix(Nstates, rate_model, min_rate=0, max_rate=1)
```

Arguments

`Nstates`	The number of distinct states represented in the transition matrix (number of rows & columns).
`rate_model`	Rate model that the transition matrix must satisfy. Can be "ER" (all rates equal), "SYM" (transition rate i–>j is equal to transition rate j–>i), "ARD" (all rates can be different) or "SUEDE" (only stepwise transitions i–>i+1 and i–>i-1 allowed, all 'up' transitions are equal, all 'down' transitions are equal).
`min_rate`	A non-negative number, specifying the minimum rate in off-diagonal entries of the transition matrix.
`max_rate`	A non-negative number, specifying the maximum rate in off-diagonal entries of the transition matrix. Must not be smaller than `min_rate`.

Value

A real-valued quadratic matrix of size Nstates x Nstates, representing a transition matrix for an Mk model. Each row will sum to 0. The [r,c]-th entry represents the transition rate r–>c. The number of unique off-diagonal rates will depend on the `rate_model` chosen.

Author(s)

Stilianos Louca

See Also

`exponentiate_matrix`, `get_stationary_distribution`

Examples

```
# generate a 5x5 Markov transition rate matrix
Q = get_random_mk_transition_matrix(Nstates=5, rate_model="ARD")
```

`get_reds` *Calculate relative evolutionary divergences in a tree.*

Description

Calculate the relative evolutionary divergence (RED) of each node in a rooted phylogenetic tree. The RED of a node is a measure of its relative placement between the root and the node's descending tips (Parks et al. 2018). The root's RED is always 0, the RED of each tip is 1, and the RED of each node is between 0 and 1.

Usage

```
get_reds(tree)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.

Details

The RED of a node measures its relative placement between the root and the node's descending tips (Parks et al. 2018). The root's RED is set to 0. Traversing from root to tips (preorder traversal), for each node the RED is set to $P + (a/(a+b)) \cdot (1-P)$, where P is the RED of the node's parent, a is the edge length connecting the node to its parent, and b is the average distance from the node to its descending tips. The RED of a tip would always be 1.

The RED may be useful for defining taxonomic ranks based on a molecular phylogeny (e.g. see Parks et al. 2018). This function is similar to the PhyloRank v0.0.27 script published by Parks et al. (2018).

The time complexity of this function is O(Nedges). The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). If `tree$edge.length` is `NULL`, then all edges in the input tree are assumed to have length 1.

Value

A numeric vector of length Nnodes, listing the RED of each node in the tree. The REDs of tips are not included, since these would always be equal to 1.

Author(s)

Stilianos Louca

References

D. H. Parks, M. Chuvochina et al. (2018). A proposal for a standardized bacterial taxonomy based on genome phylogeny. bioRxiv 256800. DOI:10.1101/256800

Examples

```
# generate a random tree
params = list(birth_rate_intercept=1, death_rate_intercept=0.8)
tree = generate_random_tree(params, max_time=100, coalescent=FALSE)$tree

# calculate and print REDs
REDs = get_reds(tree)
print(REDs)
```

`get_stationary_distribution` *Stationary distribution of Markov transition matrix.*

Description

Calculate the stationary probability distribution vector p for a transition matrix Q of a continuous-time Markov chain. That is, calculate $p \in [0,1]^n$ such that `sum(p)==0` and $p^T Q = 0$.

Usage

```
get_stationary_distribution(Q)
```

Arguments

`Q`	A valid transition rate matrix of size Nstates x Nstates, i.e. a quadratic matrix in which every row sums up to zero.

Details

A stationary distribution of a discrete-state continuous-time Markov chain is a probability distribution across states that remains constant over time, i.e. $p^T Q = 0$. Note that in some cases (i.e. if Q is not irreducible), there may be multiple distinct stationary distributions. In that case,which one is returned by this function is unpredictable. Internally, p is estimated by stepwise minimization of the norm of $p^T Q$, starting with the vector p in which every entry equals 1/Nstates.

Value

A numeric vector of size Nstates and with non-negative entries, satisfying the conditions `p%*%Q==0` and `sum(p)==1.0`.

Author(s)

Stilianos Louca

See Also

`exponentiate_matrix`

Examples

```
# generate a random 5x5 Markov transition matrix
Q = get_random_mk_transition_matrix(Nstates=5, rate_model="ARD")

# calculate stationary probability distribution
p = get_stationary_distribution(Q)
print(p)

# test correctness (p*Q should be 0, apart from rounding errors)
cat(sprintf("max(abs(p*Q)) = %g\n",max(abs(p %*% Q))))
```

---

get_subtree_at_node

Extract a subtree descending from a specific node.

Description

Given a tree and a focal node, extract the subtree descending from the focal node and place the focal node as the root of the extracted subtree.

Usage

```
get_subtree_at_node(tree, node)
```

Arguments

`tree`	A tree of class "phylo".
`node`	Character or integer specifying the name or index, respectively, of the focal node at which to extract the subtree. If an integer, it must be between 1 and `tree$Nnode`. If a character, it must be a valid entry in `tree$node.label`.

Details

The input tree need not be rooted, however "descendance" from the focal node is inferred based on the direction of edges in `tree$edge`. The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Value

A list with the following elements:

`subtree`	A new tree of class "phylo", containing the subtree descending from the focal node. This tree will be rooted, with the new root being the focal node.
`new2old_tip`	Integer vector of length Ntips_kept (=number of tips in the extracted subtree) with values in 1,..,Ntips, mapping tip indices of the subtree to tip indices in the original tree. In particular, `tree$tip.label[new2old_tip]` will be equal to `subtree$tip.label`.
`new2old_node`	Integer vector of length Nnodes_kept (=number of nodes in the extracted subtree) with values in 1,..,Nnodes, mapping node indices of the subtree to node indices in the original tree. For example, `new2old_node[1]` is the index that the first node of the subtree had within the original tree. In particular, `tree$node.label[new2old_node]` will be equal to `subtree$node.label` (if node labels are available).
`new2old_edge`	Integer vector of length Nedges_kept (=number of edges in the extracted subtree), with values in 1,..,Nedges, mapping edge indices of the subtree to edge indices in the original tree. In particular, `tree$edge.length[new2old_edge]` will be equal to `subtree$edge.length` (if edge lengths are available).

Author(s)

Stilianos Louca

See Also

`get_subtree_with_tips`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# extract subtree descending from a random node
```

```
node = sample.int(tree$Nnode,size=1)
subtree = get_subtree_at_node(tree, node)$subtree

# print summary of subtree
cat(sprintf("Subtree at %d-th node has %d tips\n",node,length(subtree$tip.label)))
```

---

get_subtree_with_tips

Extract a subtree spanning a specific subset of tips.

Description

Given a rooted tree and a subset of tips, extract the subtree containing only those tips. The root of the tree is kept.

Usage

```
get_subtree_with_tips(tree,
                      only_tips               = NULL,
                      omit_tips               = NULL,
                      collapse_monofurcations = TRUE,
                      force_keep_root         = FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`only_tips`	Either a character vector listing tip names to keep, or an integer vector listing tip indices to keep (between 1 and Ntips). Can also be `NULL`. Tips listed in `only_tips` not found in the tree will be silently ignored.
`omit_tips`	Either a character vector listing tip names to omit, or an integer vector listing tip indices to omit (between 1 and Ntips). Can also be `NULL`. Tips listed in `omit_tips` not found in the tree will be silently ignored.
`collapse_monofurcations`	A logical specifying whether nodes with a single outgoing edge remaining should be collapsed (removed). Incoming and outgoing edge of such nodes will be concatenated into a single edge, connecting the parent (or earlier) and child (or later) of the node. In that case, the returned tree will have edge lengths that reflect the concatenated edges.
`force_keep_root`	Logical, specifying whether to keep the root even if `collapse_monofurcations==TRUE` and the root of the subtree is left with a single child. If `FALSE`, and `collapse_monofurcations==T` the root may be removed and one of its descendants may become root.

Details

If both `only_tips` and `omit_tips` are `NULL`, then all tips are kept and the tree remains unchanged. If both `only_tips` and `omit_tips` are non-`NULL`, then only tips listed in `only_tips` and not listed in `omit_tips` will be kept. If `only_tips` and/or `omit_tips` is a character vector listing tip names, then `tree$tip.label` must exist.

If the input tree does not include `edge.length`, each edge in the input tree is assumed to have length 1. The root of the tree (which is always kept) is assumed to be the unique node with no incoming edge. The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

The asymptotic time complexity of this function is O(Nnodes+Ntips), where Ntips is the number of tips and Nnodes the number of nodes in the input tree.

When `only_tips==NULL`, `omit_tips!=NULL`, `collapse_monofurcations==TRUE` and `force_keep_root==FALSE`, this function is analogous to the function `drop.tip` in the `ape` package with option `trim_internal=TRUE` (v. 0.5-64).

Value

A list with the following elements:

`subtree`	A new tree of class "phylo", containing only the tips specified by `tips_to_keep` and the nodes & edges connecting those tips to the root. The returned tree will include `edge.lengh` as a member variable, listing the lengths of the remaining (possibly concatenated) edges.
`root_shift`	Numeric, indicating the phylogenetic distance between the old and the new root. Will always be non-negative.
`new2old_tip`	Integer vector of length Ntips_kept (=number of tips in the extracted subtree) with values in 1,..,Ntips, mapping tip indices of the subtree to tip indices in the original tree. In particular, `tree$tip.label[new2old_tip]` will be equal to `subtree$tip.label`.
`new2old_node`	Integer vector of length Nnodes_kept (=number of nodes in the extracted subtree) with values in 1,..,Nnodes, mapping node indices of the subtree to node indices in the original tree. For example, `new2old_node[1]` is the index that the first node of the subtree had within the original tree. In particular, `tree$node.label[new2old_node]` will be equal to `subtree$node.label` (if node labels are available).

Author(s)

Stilianos Louca

See Also

`get_subtree_at_node`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# choose a random subset of tips
tip_subset = sample.int(Ntips, size=as.integer(Ntips/10), replace=FALSE)

# extract subtree spanning the chosen tip subset
subtree = get_subtree_with_tips(tree, only_tips=tip_subset)$subtree

# print summary of subtree
cat(sprintf("Subtree has %d tips and %d nodes\n",length(subtree$tip.label),subtree$Nnode))
```

`get_tips_for_mrcas` *Find tips with specific most recent common ancestors.*

Description

Given a rooted phylogenetic tree and a list of nodes ("MRCA nodes"), for each MRCA node find a set of descending tips ("MRCA-defining tips") such that their most recent common ancestor (MRCA) is that node. This may be useful for cases where nodes need to be described as MRCAs of tip pairs for input to certain phylogenetics algorithms (e.g., for tree dating).

Usage

```
get_tips_for_mrcas(tree, mrca_nodes, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`mrca_nodes`	Either an integer vector or a character vector, listing the nodes for each of which an MRCA-defining set of tips is to be found. If an integer vector, it should list node indices (i.e. from 1 to Nnodes). If a character vector, it should list node names; in that case `tree$node.label` must exist.
`check_input`	Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.

Details

At most 2 MRCA-defining tips are assigned to each MRCA node. This function assumes that each of the `mrca_nodes` has at least two children or has a child that is a tip (otherwise the problem is not well-defined). The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child).

The asymptotic time complexity of this function is O(Ntips+Nnodes) + O(Nmrcas), where Ntips is the number of tips, Nnodes is the number of nodes in the tree and Nmrcas is equal to `length(mrca_nodes)`.

Value

A list of the same size as `mrca_nodes`, whose n-th element is an integer vector of tip indices (i.e. with values in 1,..,Ntips) whose MRCA is the n-th node listed in `mrca_nodes`.

Author(s)

Stilianos Louca

See Also

`get_pairwise_mrcas`, `get_mrca_of_set`

Examples

```
# generate a random tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1),Ntips)$tree

# pick random nodes
focal_nodes = sample.int(n=tree$Nnode, size=3, replace=FALSE)

# get tips for mrcas
tips_per_focal_node = get_tips_for_mrcas(tree, focal_nodes);

# check correctness (i.e. calculate actual MRCAs of tips)
for(n in 1:length(focal_nodes)){
  mrca = get_mrca_of_set(tree, tips_per_focal_node[[n]])
  cat(sprintf("Focal node = %d, should match mrca of tips = %d\n",focal_nodes[n],mrca-Ntips)
}
```

`get_trait_acf` *Phylogenetic autocorrelation function of a numeric trait.*

Description

Given a rooted phylogenetic tree and a numeric (typically continuous) trait with known value (state) on each tip, calculate the phylogenetic autocorrelation function (ACF) of the trait. The ACF is a function of phylogenetic distance x, where ACF(x) is the Pearson autocorrelation of the trait between two tips, provided that the tips have phylogenetic ("patristic") distance x. The function `get_trait_acf` also calculates the mean absolute difference and the mean relative difference of the trait between any two random tips at phylogenetic distance x (see details below).

Usage

```
get_trait_acf(tree, tip_states, Npairs=10000, Nbins=10)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the value of the trait at each tip in the tree. Note that tip_states[i] (where i is an integer index) must correspond to the i-th tip in the tree.
`Npairs`	Total number of random tip pairs to draw. A greater number of tip pairs will improve the accuracy of the estimated ACF within each distance bin.
`Nbins`	Number of distance bins to consider within the range of phylogenetic distances encountered between tip pairs in the tree. A greater number of bins will increase the resolution of the ACF as a function of phylogenetic distance, but will decrease the number of tip pairs falling within each bin (which reduces the accuracy of the estimated ACF).

Details

The phylogenetic autocorrelation function (ACF) of a trait can give insight into the evolutionary processes shaping its distribution across clades. An ACF that decays slowly with increasing phylogenetic distance indicates a strong phylogenetic conservatism of the trait, whereas a rapidly decaying ACF indicates weak phylogenetic conservatism. Similarly, if the mean absolute difference in trait value between two random tips increases with phylogenetic distance, this indicates a phylogenetic autocorrelation of the trait (Zaneveld et al. 2014). Here, phylogenetic distance between tips refers to their patristic distance, i.e. the minimum cumulative edge length required to connect the two tips.

Since the distances between all possible tip pairs do not cover a continuoum (as there is only a finite number of tips), this function randomly draws tip pairs from the tree, maps them onto a finite set of equally-sized distance bins and then estimates the ACF for the centroid of each distance bin based on tip pairs in that bin. In practice, as a next step one would usually plot the estimated ACF (returned vector `autocorrelations`) over the centroids of the distance bins (returned vector `distances`).

The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child). If `tree$edge.length` is missing, then every edge is assumed to have length 1. The input tree must be rooted at some node for technical reasons (see function `root_at_node`), but the choice of the root node does not influence the result.

Value

A list with the following elements:

`distances`	Numeric vector of size Nbins, storing the centroid phylogenetic distance of each distance bin in increasing order. The first and last distance bin approximately span the full range of phylogenetic distances encountered between any two random tips in the tree.
`autocorrelations`	Numeric vector of size Nbins, storing the estimated Pearson autocorrelation of the trait for each distance bin.

`mean_abs_differences`
: Numeric vector of size Nbins, storing the mean absolute difference of the trait between two random tips in each distance bin.

`mean_rel_differences`
: Numeric vector of size Nbins, storing the mean relative difference of the trait between two random tips in each distance bin. The relative difference between two values X and Y is 0 if $X == Y$, and equal to

$$\frac{|X-Y|}{0.5\cdot(|X|+|Y|)}$$

otherwise.

`Npairs_per_distance`
: Integer vector of size Nbins, storing the number of random tip pairs associated with each distance bin.

Author(s)

Stilianos Louca

References

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`get_trait_depth`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=10000)$tree

# simulate continuous trait evolution on the tree
tip_states = simulate_bm_model(tree, diffusivity=1)$tip_states

# calculate autocorrelation function
ACF = get_trait_acf(tree, tip_states, Npairs=1e7, Nbins=20)

# plot ACF (autocorrelation vs phylogenetic distance)
plot(ACF$distances, ACF$autocorrelations, type="l", xlab="distance", ylab="ACF")
```

get_trait_depth *Calculate depth of phylogenetic conservatism for a binary trait.*

Description

Given a rooted phylogenetic tree and presences/absences of a binary trait for each tip, calculate the mean phylogenetic depth at which the trait is conserved across clades, in terms of the consenTRAIT metric introduced by Martiny et al (2013). This is the mean depth of clades that are positive in the trait (i.e. in which a sufficient fraction of tips exhibits the trait).

Usage

```
get_trait_depth(tree,
                tip_states,
                min_fraction        = 0.9,
                count_singletons    = TRUE,
                singleton_resolution= 0,
                weighted            = FALSE,
                Npermutations       = 0)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips indicating absence (value <=0) or presence (value >0) of a particular trait at each tip of the tree. Note that tip_states[i] (where i is an integer index) must correspond to the i-th tip in the tree.
`min_fraction`	Minimum fraction of tips in a clade exhibiting the trait, for the clade to be considered "positive" in the trait. In the original paper by Martiny et al (2013), this was 0.9.
`count_singletons`	Logical, specifying whether to include singletons in the statistics (tips positive in the trait, but not part of a larger positive clade). The phylogenetic depth of singletons is taken to be half the length of their incoming edge, as proposed by Martiny et al (2013). If FALSE, singletons are ignored.
`singleton_resolution`	Numeric, specifying the phylogenetic resolution at which to resolve singletons. Any clade found to be positive in a trait will be considered a singleton if the distance of the clade's root to all descending tips is below this threshold.
`weighted`	Whether to weight positive clades by their number of positive tips. If FALSE, each positive clades is weighted equally, as proposed by Martiny et al (2013).
`Npermutations`	Number of random permutations for estimating the statistical significance of the mean trait depth. If zero (default), the statistical significance is not calculated.

Details

This function calculates the "consenTRAIT" metric (or variants thereof) proposed by Martiny et al. (2013) for measuring the mean phylogenetic depth at which a binary trait (e.g. presence/absence of a particular metabolic function) is conserved across clades. A greater mean depth means that the trait tends to be conserved in deeper-rooting clades. In their original paper, Martiny et al. proposed to consider a trait as conserved in a clade (i.e. marking a clade as "positive" in the trait) if at least 90% of the clade's tips exhibit the trait (i.e. are "positive" in the trait). This fraction can be controlled using the `min_fraction` parameter. The depth of a clade is taken as the average distance of its tips to the clade's root.

The default parameters of this function reflect the original choices made by Martiny et al. (2013), however in some cases it may be sensible to adjust them. For example, if you suspect a high risk of false positives in the detection of a trait, it may be worth setting `count_singletons` to `FALSE` to avoid skewing the distribution of conservation depths towards shallower depths due to false positives.

The statistical significance of the calculated mean depth, i.e. the probability of encountering such a mean dept or higher by chance, can be estimated based on a null model in which each tip is randomly and independently re-assigned a presence or absence of the trait. In the null model, the probability that a tip exhibits the trait is set to the fraction of positive entries in `tip_states`.

The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). If `tree$edge.length` is missing, then every edge is assumed to have length 1.

Value

A list with the following elements:

`mean_depth`	Mean phylogenetic depth of clades that are positive in the trait.
`var_depth`	Variance of phylogenetic depths of clades that are positive in the trait.
`min_depth`	Minimum phylogenetic depth of clades that are positive in the trait.
`max_depth`	Maximum phylogenetic depth of clades that are positive in the trait.
`Npositives`	Number of clades that are positive in the trait.
`P`	Statistical significance (P-values) of mean_depth, under a null model of random trait presences/absences (see details above). This is the probability that under the null model, the mean_depth would be at least as high as observed in the data.
`mean_random_depth`	Mean random mean_depth, under a null model of random trait presences/absences (see details above).
`positive_clades`	Integer vector, listing indices of tips and nodes (from 1 to Ntips+Nnodes) that were found to be positive in the trait and counted towards the statistic.
`positives_per_clade`	Integer vector of size Ntips+Nnodes, listing the number of descending tips per clade (tip or node) that were positive in the trait.
`mean_depth_per_clade`	Numeric vector of size Ntips+Nnodes, listing the mean phylogenetic depth of each clade (tip or node), i.e. the average distance to all its descending tips.

Author(s)

Stilianos Louca

References

A. C. Martiny, K. Treseder and G. Pusch (2013). Phylogenetic trait conservatism of functional traits in microorganisms. ISME Journal. 7:830-838.

See Also

`get_trait_acf`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=1000)$tree

# simulate binary trait evolution on the tree
Q = get_random_mk_transition_matrix(Nstates=2, rate_model="ARD", max_rate=0.1)
tip_states = simulate_mk_model(tree, Q)$tip_states

# change states from 1/2 to 0/1 (presence/absence)
tip_states = tip_states - 1

# calculate phylogenetic conservatism of trait
results = get_trait_depth(tree, tip_states, count_singletons=FALSE, weighted=TRUE)
cat(sprintf("Mean depth = %g, std = %g\n",results$mean_depth,sqrt(results$var_depth)))
```

`get_trait_stats_over_time`

Calculate mean & standard deviation of a numeric trait on a dated tree over time.

Description

Given a rooted and dated phylogenetic tree, and a scalar numeric trait with known value on each node and tip of the tree, calculate the mean and the variance of the trait's states across the tree at discrete time points. For example, if the trait represents "body size", then this function calculates the mean body size of extant clades over time.

Usage

```
get_trait_stats_over_time(tree, states, Ntimes=NULL, times=NULL, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo", where edge lengths represent time intervals (or similar).
`states`	Numeric vector, specifying the trait's state at each tip and each node of the tree (in the order in which tips & nodes are indexed). May include `NA` or `NaN` if values are missing for some tips/nodes.
`Ntimes`	Integer, number of equidistant time points for which to calculade clade counts. Can also be `NULL`, in which case `times` must be provided.
`times`	Integer vector, listing time points (in ascending order) for which to calculate clade counts. Can also be `NULL`, in which case `Ntimes` must be provided.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

If `tree$edge.length` is missing, then every edge in the tree is assumed to be of length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The tree need not be ultrametric (e.g. may include extinct tips), although in general this function only makes sense if edge lengths correspond to time (or similar).

Either `Ntimes` or `times` must be non-`NULL`, but not both. `states` need not include names; if it does, then these are checked to be in the same order as in the tree (if `check_input==TRUE`).

Value

A list with the following elements:

`Ntimes`	Integer, indicating the number of returned time points. Equal to the provided `Ntimes` if applicable.
`times`	Numeric vector of size Ntimes, listing the considered time points in increasing order. If `times` was provided as an argument to the function, then this will be the same as provided.
`clade_counts`	Integer vector of size Ntimes, listing the number of tips or nodes considered at each time point.
`means`	Numeric vector of size Ntimes, listing the arithmetic mean of trait states at each time point.
`stds`	Numeric vector of size Ntimes, listing the population standard deviation of trait states at each time point.

Author(s)

Stilianos Louca

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1), max_tips=1000)$tree

# simulate a numeric trait under Brownian-motion
trait = simulate_bm_model(tree, diffusivity=1)
states = c(trait$tip_states,trait$node_states)

# calculate trait stats over time
results = get_trait_stats_over_time(tree, states, Ntimes=100)

# plot trait stats over time (mean +/- std)
M = results$means
S = results$stds
matplot(x=results$times,
        y=matrix(c(M-S,M+S),ncol=2,byrow=FALSE),
        main = "Simulated BM trait over time",
        lty = 1, col="black",
        type="l", xlab="time", ylab="mean +/- std")
```

`get_transition_index_matrix`

Create an index matrix for a Markov transition model.

Description

Create an index matrix encoding the parametric structure of the transition rates in a discrete-state continuous-time Markov model (e.g., Mk model of trait evolution). Such an index matrix is required by certain functions for mapping independent rate parameters to transition rates. For example, an index matrix may encode the information that each rate i–>j is equal to its reversed counterpart j–>i.

Usage

```
get_transition_index_matrix(Nstates, rate_model)
```

Arguments

`Nstates`	Integer, the number of distinct states represented in the transition matrix (number of rows & columns).
`rate_model`	Rate model that the transition matrix must satisfy. Can be "ER" (all rates equal), "SYM" (transition rate i–>j is equal to transition rate j–>i), "ARD" (all rates can be different) or "SUEDE" (only stepwise transitions i–>i+1 and i–>i-1 allowed, all 'up' transitions are equal, all 'down' transitions are equal).

Details

The returned index matrix will include as many different positive integers as there are independent rate parameters in the requested rate model, plus potentially the value 0 (which has a special meaning, see below).

Value

A named list with the following elements:

`index_matrix`	Integer matrix of size Nstates x Nstates, with values between 0 and Nstates, assigning each entry in the transition matrix to an independent transition rate parameter. A value of 0 means that the corresponding rate is fixed to zero (if off-diagonal) or will be adjusted to ensure a valid Markov transition rate matrix (if on the diagonal).
`Nrates`	Integer, the number of independent rate parameters in the model.

Author(s)

Stilianos Louca

See Also

`get_random_mk_transition_matrix`

`get_tree_span` *Get min and max distance of any tip to the root.*

Description

Given a rooted phylogenetic tree, calculate the minimum and maximum phylogenetic distance (cumulative branch length) of any tip from the root.

Usage

```
get_tree_span(tree, as_edge_count=FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`as_edge_count`	Logical, specifying whether distances should be counted in number of edges, rather than cumulative edge length. This is the same as if all edges had length 1.

Details

If `tree$edge.length` is missing, then every edge in the tree is assumed to be of length 1. The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child). The asymptotic average time complexity of this function is O(Nedges), where Nedges is the number of edges in the tree.

Value

A named list with the following elements:

`min_distance`	Minimum phylogenetic distance that any of the tips has to the root.
`max_distance`	Maximum phylogenetic distance that any of the tips has to the root.

Author(s)

Stilianos Louca

See Also

`get_pairwise_distances`

Examples

```
# generate a random tree
Ntips   = 1000
params  = list(birth_rate_intercept=1, death_rate_intercept=0.5)
tree    = generate_random_tree(params, max_tips=Ntips, coalescent=FALSE)$tree

# calculate min & max tip distances from root
tree_span = get_tree_span(tree)
cat(sprintf("Tip min dist = %g, max dist = %g\n",
            tree_span$min_distance,
            tree_span$max_distance))
```

`get_tree_traversal_root_to_tips`
Traverse tree from root to tips.

Description

Create data structures for traversing a tree from root to tips, and for efficient retrieval of a node's outgoing edges and children.

Usage

```
get_tree_traversal_root_to_tips(tree, include_tips)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`include_tips`	Include tips in the tarversal queue. If FALSE, then only nodes are included in the queue.

Details

Many dynamic programming algorithms for phylogenetics involve traversing the tree in a certain direction (root to tips or tips to root), and efficient (O(1) complexity) access to a node's direct children can significantly speed up those algorithms. This function is meant to provide data structures that allow traversing the tree's nodes (and optionally tips) in such an order that each node is traversed prior to its descendants (root–>tips) or such that each node is traversed after its descendants (tips–>root). This function is mainly meant for use in other algorithms, and is probably of little relevance to the average user.

The tree may include multi-furcations as well as mono-furcations (i.e. nodes with only one child).

The asymptotic time and memory complexity of this function is O(Ntips), where Ntips is the number of tips in the tree.

Value

A list with the following elements:

`queue`	An integer vector of size Nnodes (if `include_tips` was `FALSE`) or of size Nnodes+Ntips (if `include_tips` was `TRUE`), listing indices of nodes (and optionally tips) in the order root–>tips described above. In particular, `queue[1]` will be the index of the tree's root (typically Ntips+1).
`edges`	An integer vector of size Nedges (=`nrow(tree$edge)`), listing indices of edges (corresponding to `tree$edge`) such that outgoing edges of the same node are listed in consequtive order.
`node2first_edge`	An integer vector of size Nnodes listing the location of the first outgoing edge of each node in `edges`. That is, `edges[node2first_edge[n]]` points to the first outgoing edge of node n in `tree$edge`.
`node2last_edge`	An integer vector of size Nnodes listing the location of the last outgoing edge of each node in `edges`. That is, `edges[node2last_edge[n]]` points to the last outgoing edge of node n in `tree$edge`. The total number of outgoing edges of a node is thus given by `1+node2last_edge[n]-node2first_edge[n]`.

Author(s)

Stilianos Louca

See Also

`reorder_tree_edges`

Examples

```
## Not run:
get_tree_traversal_root_to_tips(tree, include_tips=TRUE)

## End(Not run)
```

`hsp_empirical_probabilities` *Hidden state prediction via empirical probabilities.*

Description

Reconstruct ancestral discrete states of nodes and predict unknown (hidden) states of tips on a tree based on empirical state probabilities across tips. This is a very crude HSP method, and other more sophisticated methods should be preferred (e.g. `hsp_mk_model`).

Usage

```
hsp_empirical_probabilities(tree, tip_states,
                            Nstates=NULL, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree as an integer from 1 to Nstates, where Nstates is the possible number of states (see below). `tip_states` can include `NA` to indicate an unknown tip state that is to be predicted.
`Nstates`	Either `NULL`, or an integer specifying the number of possible states of the trait. If `NULL`, then it will be computed based on the maximum non-`NA` value encountered in `tip_states`
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

Any `NA` entries in `tip_states` are interpreted as unknown states. Prior to ancestral state reconstruction, the tree is temporarily prunned, keeping only tips with known state. The function then calculates the empirical state probabilities for each node in the pruned tree, based on the states across tips descending from each node. The state probabilities of tips with unknown state are set to those of the most recent ancestor with reconstructed states, as described by Zaneveld and Thurber (2014).

The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). This function has asymptotic time complexity O(Nedges x Nstates).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

This function is meant for reconstructing ancestral states in all nodes of a tree as well as predicting the states of tips with an a priory unknown state. If the state of all tips is known and only ancestral state reconstruction is needed, consider using functions such as `asr_empirical_probabilities` for improved efficiency.

Value

A list with the following elements:

`success`	Logical, indicating whether HSP was successful. If `FALSE`, some return values may be `NULL`.
`likelihoods`	A 2D numeric matrix, listing the probability of each tip and node being in each state. This matrix will have (Ntips+Nnodes) rows and Nstates columns, where Nstates was either explicitly provided as an argument or inferred based on the number of unique values in `tip_states` (if `Nstates` was passed as NULL). In the latter case, the column names of this matrix will be the unique values found in `tip_states`. The rows in this matrix will be in the order in which tips and nodes are indexed in the tree, i.e. the rows 1,..,Ntips store the probabilities for tips, while rows (Ntips+1),..,(Ntips+Nnodes) store the probabilities for nodes. Each row in this matrix will sum up to 1. Note that the return value is named this way for compatibility with other HSP functions.

Author(s)

Stilianos Louca

References

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`hsp_max_parsimony`, `hsp_mk_model`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a discrete trait
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ER", max_rate=0.1)
tip_states = simulate_mk_model(tree, Q)$tip_states
```

```
# print states of first 20 tips
print(tip_states[1:20])

# set half of the tips to unknown state
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via MPR
likelihoods = hsp_empirical_probabilities(tree, tip_states, Nstates)$likelihoods
estimated_tip_states = max.col(likelihoods[1:Ntips,])

# print estimated states of first 20 tips
print(estimated_tip_states[1:20])
```

---

hsp_independent_contrasts
Hidden state prediction via phylogenetic independent contrasts.

Description

Reconstruct ancestral states of a continuous (numeric) trait for nodes and predict unknown (hidden) states for tips on a tree using phylogenetic independent contrasts.

Usage

```
hsp_independent_contrasts(tree, tip_states, weighted=TRUE, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the state of each tip in the tree. `tip_states` can include `NA` to indicate an unknown tip state that is to be predicted.
`weighted`	Logical, specifying whether to weight transition costs by the inverted edge lengths during ancestral state reconstruction. This corresponds to the "weighted squared-change parsimony" reconstruction by Maddison (1991) for a Brownian motion model of trait evolution.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

Any `NA` entries in `tip_states` are interpreted as unknown (hidden) states to be estimated. Prior to ancestral state reconstruction, the tree is temporarily prunned, keeping only tips with known state. The function then uses a postorder traversal algorithm to calculate the intermediate "X" variables (a state estimate for each node) introduced by Felsenstein (1985) in his phylogenetic independent contrasts method. Note that these are only local estimates, i.e. for each node the estimate is only based on the tip states in the subtree descending from that node (see discussion in Garland and Ives, 2000). The states of tips with hidden state are set to those of the most recent ancestor with reconstructed state, as described by Zaneveld and Thurber (2014).

This function has asymptotic time complexity O(Nedges). If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. This is the same as setting `weighted=FALSE`. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

This function is meant for reconstructing ancestral states in all nodes of a tree as well as predicting the states of tips with an a priory unknown state. If the state of all tips is known and only ancestral state reconstruction is needed, consider using the function `asr_independent_contrasts` for improved efficiency.

Value

A list with the following elements:

`success`	Logical, indicating whether HSP was successful. If `FALSE`, some return values may be `NULL`.
`states`	A numeric vector of size Ntips+Nnodes, listing the reconstructed state of each tip and node. The entries in this vector will be in the order in which tips and nodes are indexed in `tree$edge`.
`total_sum_of_squared_changes`	The total sum of squared changes in tree, minimized by the (optionally weighted) squared-change parsimony algorithm. This is equation 7 in (Maddison, 1991). Note that for the root, phylogenetic independent contrasts is equivalent to Maddison's squared-change parsimony.

Author(s)

Stilianos Louca

References

J. Felsenstein (1985). Phylogenies and the comparative method. The American Naturalist. 125:1-15.

T. Jr. Garland and A. R. Ives (2000). Using the past to predict the present: Confidence intervals for regression equations in phylogenetic comparative methods. The American Naturalist. 155:346-364.

W. P. Maddison (1991). Squared-change parsimony reconstructions of ancestral states for continuous-valued characters on a phylogenetic tree. Systematic Zoology. 40:304-314.

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`asr_squared_change_parsimony` `hsp_max_parsimony`, `hsp_mk_model`,

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# print tip states
print(as.vector(tip_states))

# set half of the tips to unknown state
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via weighted PIC
estimated_states = hsp_independent_contrasts(tree, tip_states, weighted=TRUE)$states

# print estimated tip states
print(estimated_states[1:Ntips])
```

`hsp_max_parsimony` *Hidden state prediction via maximum parsimony.*

Description

Reconstruct ancestral discrete states of nodes and predict unknown (hidden) states of tips on a tree using maximum parsimony. Transition costs can vary between transitions, and can optionally be weighted by edge length.

Usage

```
hsp_max_parsimony(tree, tip_states, Nstates=NULL,
                  transition_costs="all_equal",
                  edge_exponent=0.0, weight_by_scenarios=TRUE,
                  check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree as an integer from 1 to Nstates, where Nstates is the possible number of states (see below). `tip_states` can include `NA` to indicate an unknown tip state that is to be predicted.

`Nstates`	Either `NULL`, or an integer specifying the number of possible states of the trait. If `NULL`, then it will be computed based on the maximum non-`NA` value encountered in `tip_states`
`transition_costs`	Same as for the function `asr_max_parsimony`.
`edge_exponent`	Same as for the function `asr_max_parsimony`.
`weight_by_scenarios`	Logical, indicating whether to weight each optimal state of a node by the number of optimal maximum-parsimony scenarios in which the node is in that state. If FALSE, then all possible states of a node are weighted equally (i.e. are assigned equal probabilities).
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. The order of states (if relevant) should be reflected in their integer representation. For example, if your original states are "small", "medium" and "large" and `transition_costs=="sequential"`, it is advised to represent these states as integers 1,2,3. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

Any `NA` entries in `tip_states` are interpreted as unknown states. Prior to ancestral state reconstruction, the tree is temporarily prunned, keeping only tips with known state. The function then applies Sankoff's (1975) dynamic programming algorithm for ancestral state reconstruction, which determines the smallest number (or least costly if transition costs are uneven) of state changes along edges needed to reproduce the known tip states. The state probabilities of tips with unknown state are set to those of the most recent ancestor with reconstructed states, as described by Zaneveld and Thurber (2014). This function has asymptotic time complexity O(Ntips+Nnodes x Nstates).

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. If `edge_exponent` is 0, then edge lengths do not influence the result. If `edge_exponent!=0`, then all edges must have non-zero length. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. None of the input vectors or matrixes need include row or column names; if they do, however, they are checked for consistency (if `check_input==TRUE`).

This function is meant for reconstructing ancestral states in all nodes of a tree as well as predicting the states of tips with an a priory unknown state. If the state of all tips is known and only ancestral state reconstruction is needed, consider using the function `asr_max_parsimony` for improved efficiency.

Value

A list with the following elements:

`success`	Logical, indicating whether HSP was successful. If `FALSE`, some return values may be `NULL`.
`likelihoods`	A 2D numeric matrix, listing the probability of each tip and node being in each state. This matrix will have (Ntips+Nnodes) rows and Nstates columns, where Nstates was either explicitly provided as an argument or inferred based on the number of unique values in `tip_states` (if `Nstates` was passed as NULL). In the latter case, the column names of this matrix will be the unique values found in `tip_states`. The rows in this matrix will be in the order in which tips and nodes are indexed in the tree, i.e. the rows 1,..,Ntips store the probabilities for tips, while rows (Ntips+1),..,(Ntips+Nnodes) store the probabilities for nodes. Each row in this matrix will sum up to 1. Note that the return value is named this way for compatibility with other HSP functions.

Author(s)

Stilianos Louca

References

D. Sankoff (1975). Minimal mutation trees of sequences. SIAM Journal of Applied Mathematics. 28:35-42.

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

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`asr_max_parsimony`, `asr_mk_model`, `hsp_mk_model`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 10
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a discrete trait
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ER")
tip_states = simulate_mk_model(tree, Q)$tip_states

# print tip states
print(tip_states)

# set half of the tips to unknown state
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via MPR
likelihoods = hsp_max_parsimony(tree, tip_states, Nstates)$likelihoods
estimated_tip_states = max.col(likelihoods[1:Ntips,])
```

```
# print estimated tip states
print(estimated_tip_states)
```

`hsp_mk_model` *Hidden state prediction with Mk models and rerooting*

Description

Reconstruct ancestral states of a discrete trait and predict unknown (hidden) states of tips using a fixed-rates continuous-time Markov model (a.k.a. "Mk model"). This function can fit the model (i.e. estimate the transition matrix) using maximum likelihood, or use a specified transition matrix. The function can optionally calculate marginal ancestral state likelihoods for each node in the tree, using the rerooting method by Yang et al. (1995). A subset of the tips may have completely unknown states; in this case the fitted Markov model is used to predict their state likelihoods based on their most recent reconstructed ancestor, as described by Zaneveld and Thurber (2014). The function can account for biases in which tips have known state (“reveal bias”).

Usage

```
hsp_mk_model( tree,
              tip_states,
              Nstates = NULL,
              reveal_fractions = NULL,
              tip_priors = NULL,
              rate_model = "ER",
              transition_matrix = NULL,
              include_likelihoods = TRUE,
              root_prior = "empirical",
              Ntrials = 1,
              optim_algorithm = "nlminb",
              optim_max_iterations = 200,
              optim_rel_tol = 1e-8,
              store_exponentials = TRUE,
              check_input = TRUE,
              Nthreads = 1)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	An integer vector of size Ntips, specifying the state of each tip in the tree in terms of an integer from 1 to Nstates, where Nstates is the possible number of states (see below). Can also be `NULL`, in which case `tip_priors` must not be `NULL` (see below). `tip_states` can include `NA` to indicate an unknown (hidden) tip state that is to be predicted.

`Nstates`	Either NULL, or an integer specifying the number of possible states of the trait. If `Nstates==NULL`, then it will be computed based on the maximum non-`NA` value encountered in `tip_states` or based on the number of columns in `tip_priors` (whichever is non-`NULL`).
`reveal_fractions`	Either NULL, or an integer vector of size Nstates, specifying the fraction of tips with revealed (i.e., non-hidden) state, depending on the tip state. That is, `reveal_fractions[s]` is the probability that a given tip at state `s` will have known (i.e., non-hidden) state, conditional upon being included in the tree. If the tree only contains a random subset of species (sampled independently of each species' state), then `reveal_fractions[s]` is the probability of knowing the state of a species (regardless of whether it is included in the tree), if its state is `s`. This variable can be used to account for biases in which tips have known state, depending on their state. Only the relative ratios among reveal fractions matter, i.e. multiplying `reveal_fractions` with a constant factor has no effect.
`tip_priors`	A 2D numeric matrix of size Ntips x Nstates, where Nstates is the possible number of states for the character modelled. Can also be `NULL`. Each row of this matrix must be a probability vector, i.e. it must only contain non-negative entries and must sum up to 1. The [i,s]-th entry should be the prior probability of tip i being in state s. If you know for certain that tip i is in some state s, you can set the corresponding entry to 1 and all other entries in that row to 0. A row can include `NA` to indicate that neither the state nor the probability distribution of a state are known for that tip. If for all tips you either know the exact state or have no information at all, you can also use `tip_states` instead. If `tip_priors==NULL`, then `tip_states` must not be `NULL` (see above).
`rate_model`	Rate model to be used for fitting the transition rate matrix. Similar to the `rate_model` option in the function `asr_mk_model`. See the details of `asr_mk_model` on the assumptions of each `rate_model`.
`transition_matrix`	Either a numeric quadratic matrix of size Nstates x Nstates containing fixed transition rates, or `NULL`. The [r,c]-th entry in this matrix should store the transition (probability) rate from the state r to state c. Each row in this matrix must have sum zero. If `NULL`, then the transition rates will be estimated using maximum likelihood, based on the `rate_model` specified.
`include_likelihoods`	Boolean, specifying whether to include the marginal state likelihoods for all tips and nodes, as returned variables. Setting this to `TRUE` can substantially increase computation time. If `FALSE`, the Mk model is merely fitted, but ancestral states and hidden tip states are not reconstructed.
`root_prior`	Prior probability distribution of the root's states. Similar to the `root_prior` option in the function `asr_mk_model`.
`Ntrials`	Number of trials (starting points) for fitting the transition matrix. Only relevant if `transition_matrix=NULL`. A higher number may reduce the risk of landing in a local non-global optimum of the likelihood function, but will increase computation time during fitting.

`optim_algorithm`
: Either "optim" or "nlminb", specifying which optimization algorithm to use for maximum-likelihood estimation of the transition matrix. Only relevant if `transition_matrix==NULL`.

`optim_max_iterations`
: Maximum number of iterations (per fitting trial) allowed for optimizing the likelihood function.

`optim_rel_tol`
: Relative tolerance (stop criterion) for optimizing the likelihood function.

`store_exponentials`
: Logical, specifying whether to pre-calculate and store exponentials of the transition matrix during calculation of ancestral likelihoods. This may reduce computation time because each exponential is only calculated once, but will use up more memory since all exponentials are stored. Only relevant if `include_ancestral_likelihood` is `TRUE`, otherwise exponentials are never stored.

`check_input`
: Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

`Nthreads`
: Number of parallel threads to use for running multiple fitting trials simultaneously. This only makes sense if your computer has multiple cores/CPUs and `Ntrials>1`, and is only relevant if `transition_matrix==NULL`.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. Note that Nstates can be chosen to be larger than the number of states observed in the tips of the present tree, to account for potential states not yet observed. If the trait's states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. The order of states (if applicable) should be reflected in their integer representation. For example, if your original states are "small", "medium" and "large" and `rate_model=="SUEDE"`, it is advised to represent these states as integers 1,2,3. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

This function allows the specification of the precise tip states (if these are known) using the vector `tip_states`. Alternatively, if some tip states are only known in terms of a probability distribution, you can pass these probability distributions using the matrix `tip_priors`. Note that exactly one of the two arguments, `tip_states` or `tip_priors`, must be non-`NULL`. In either case, the presence of `NA` in `tip_states` or in a row of `tip_priors` is interpreted as an absence of information about the tip's state (i.e. the tip has "hidden state").

Tips must be represented in `tip_states` or `tip_priors` in the same order as in `tree$tip.label`. None of the input vectors or matrixes need include row or column names; if they do, however, they are checked for consistency (if `check_input==TRUE`).

This method assumes that the tree is either complete (i.e. includes all species), or that the tree's tips represent a random subset of species that have been sampled independent of their state. The function does not require that tip state knowledge is independent of tip state, provided that the associated biases are known (provided via `reveal_fractions`). The rerooting method by Yang et al (2015) is used to reconstruct the marginal ancestral state likelihoods for each node by treating the node as a root and calculating its conditional scaled likelihoods. The state likelihoods of tips with hidden states are calculated from those of the most recent ancestor with previously calculated state likelihoods, using the exponentiated transition matrix along the connecting edges (essentially using the rerooting method).

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Value

A list with the following elements:

`success`	Logical, indicating whether HSP was successful. If `FALSE`, some return values may be `NULL`.
`Nstates`	Integer, specifying the number of modeled trait states.
`transition_matrix`	A numeric quadratic matrix of size Nstates x Nstates, containing the transition rates of the Markov model. The [r,c]-th entry is the transition rate from state r to state c. Will be the same as the input `transition_matrix`, if the latter was not `NULL`.
`loglikelihood`	Log-likelihood of the Markov model. If `transition_matrix` was `NULL` in the input, then this will be the log-likelihood maximized during fitting.
`likelihoods`	A 2D numeric matrix, listing the probability of each tip and node being in each state. Only included if `include_likelihoods` was `TRUE`. This matrix will have (Ntips+Nnodes) rows and Nstates columns, where Nstates was either explicitly provided as an argument, or inferred from `tip_states` or `tip_priors` (whichever was non-`NULL`). The rows in this matrix will be in the order in which tips and nodes are indexed in the tree, i.e. rows 1,..,Ntips store the probabilities for tips, while rows (Ntips+1),..,(Ntips+Nnodes) store the probabilities for nodes. For example, `likelihoods[1,3]` will store the probability that tip 1 is in state 3. Each row in this matrix will sum up to 1.

Author(s)

Stilianos Louca

References

Z. Yang, S. Kumar and M. Nei (1995). A new method for inference of ancestral nucleotide and amino acid sequences. Genetics. 141:1641-1650.

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`hsp_max_parsimony`, `hsp_squared_change_parsimony`, `asr_mk_model`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 1000
tree  = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a discrete trait
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ER", max_rate=0.01)
tip_states = simulate_mk_model(tree, Q)$tip_states
cat(sprintf("Simulated ER transition rate=%g\n",Q[1,2]))

# print states for first 20 tips
print(tip_states[1:20])

# set half of the tips to unknown state
# chose tips randomly, regardless of their state (no biases)
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via Mk model max-likelihood
results = hsp_mk_model(tree, tip_states, Nstates, rate_model="ER", Ntrials=2, Nthreads=2)
estimated_tip_states = max.col(results$likelihoods[1:Ntips,])

# print Mk model fitting summary
cat(sprintf("Mk model: log-likelihood=%g\n",results$loglikelihood))
cat(sprintf("Universal (ER) transition rate=%g\n",results$transition_matrix[1,2]))

# print estimated states for first 20 tips
print(estimated_tip_states[1:20])
```

`hsp_squared_change_parsimony` *Hidden state prediction via squared-change parsimony.*

Description

Reconstruct ancestral states of a continuous (numeric) trait for nodes and predict unknown (hidden) states for tips on a tree using squared-change (or weighted squared-change) parsimony (Maddison 1991).

Usage

```
hsp_squared_change_parsimony(tree, tip_states, weighted=TRUE, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.

`tip_states`	A numeric vector of size Ntips, specifying the state of each tip in the tree. `tip_states` can include `NA` to indicate an unknown tip state that is to be predicted.
`weighted`	Logical, specifying whether to weight transition costs by the inverted edge lengths during ancestral state reconstruction. This corresponds to the "weighted squared-change parsimony" reconstruction by Maddison (1991) for a Brownian motion model of trait evolution.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

Any `NA` entries in `tip_states` are interpreted as unknown (hidden) states to be estimated. Prior to ancestral state reconstruction, the tree is temporarily prunned, keeping only tips with known state. The function then uses Maddison's squared-change parsimony algorithm to reconstruct the globally parsimonious state at each node (Maddison 1991). The states of tips with hidden state are set to those of the most recent ancestor with reconstructed state, as described by Zaneveld and Thurber (2014). This function has asymptotic time complexity O(Nedges). If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. This is the same as setting `weighted=FALSE`. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child).

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`).

This function is meant for reconstructing ancestral states in all nodes of a tree as well as predicting the states of tips with an a priory unknown state. If the state of all tips is known and only ancestral state reconstruction is needed, consider using the function `asr_squared_change_parsimony` for improved efficiency.

Value

A list with the following elements:

`states`	A numeric vector of size Ntips+Nnodes, listing the reconstructed state of each tip and node. The entries in this vector will be in the order in which tips and nodes are indexed in `tree$edge`.
`total_sum_of_squared_changes`	The total sum of squared changes, minimized by the (optionally weighted) squared-change parsimony algorithm. This is equation 7 in (Maddison, 1991).

Author(s)

Stilianos Louca

References

W. P. Maddison (1991). Squared-change parsimony reconstructions of ancestral states for continuous-valued characters on a phylogenetic tree. Systematic Zoology. 40:304-314.

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

`asr_squared_change_parsimony` `hsp_max_parsimony`, `hsp_mk_model`, `map_to_state_space`

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# print tip states
print(tip_states)

# set half of the tips to unknown state
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via weighted SCP
estimated_states = hsp_squared_change_parsimony(tree, tip_states, weighted=TRUE)$states

# print estimated tip states
print(estimated_states[1:Ntips])
```

`hsp_subtree_averaging` *Hidden state prediction via subtree averaging.*

Description

Reconstruct ancestral states of a continuous (numeric) trait for nodes and predict unknown (hidden) states for tips on a tree using subtree averaging.

Usage

```
hsp_subtree_averaging(tree, tip_states, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`tip_states`	A numeric vector of size Ntips, specifying the state of each tip in the tree. `tip_states` can include `NA` to indicate an unknown tip state that is to be predicted.
`check_input`	Logical, specifying whether to perform some basic checks on the validity of the input data. If you are certain that your input data are valid, you can set this to `FALSE` to reduce computation.

Details

Any `NA` entries in `tip_states` are interpreted as unknown (hidden) states to be estimated. For each node the reconstructed state is set to the arithmetic average state of all tips with known state and descending from that node. For each tip with hidden state and each node whose descending tips all have hidden states, the state is set to the state of the closest ancestral node with known or reconstructed state, while traversing from root to tips (Zaneveld and Thurber 2014). Note that reconstructed node states are only local estimates, i.e. for each node the estimate is only based on the tip states in the subtree descending from that node.

Tips must be represented in `tip_states` in the same order as in `tree$tip.label`. The vector `tip_states` need not include item names; if it does, however, they are checked for consistency (if `check_input==TRUE`). This function has asymptotic time complexity O(Nedges).

This function is meant for reconstructing ancestral states in all nodes of a tree as well as predicting the states of tips with an a priory unknown state. If the state of all tips is known and only ancestral state reconstruction is needed, consider using the function `asr_subtree_averaging` for improved efficiency.

Value

A list with the following elements:

`success`	Logical, indicating whether HSP was successful.
`states`	A numeric vector of size Ntips+Nnodes, listing the reconstructed state of each tip and node. The entries in this vector will be in the order in which tips and nodes are indexed in `tree$edge`.

Author(s)

Stilianos Louca

References

J. R. Zaneveld and R. L. V. Thurber (2014). Hidden state prediction: A modification of classic ancestral state reconstruction algorithms helps unravel complex symbioses. Frontiers in Microbiology. 5:431.

See Also

asr_subtree_averaging, hsp_squared_change_parsimony

Examples

```
# generate random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# simulate a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=0, spread=1, decay_rate=0.001)$tip_stat

# print tip states
print(as.vector(tip_states))

# set half of the tips to unknown state
tip_states[sample.int(Ntips,size=as.integer(Ntips/2),replace=FALSE)] = NA

# reconstruct all tip states via subtree averaging
estimated_states = hsp_subtree_averaging(tree, tip_states)$states

# print estimated tip states
print(estimated_states[1:Ntips])
```

is_monophyletic   *Determine if a set of tips is monophyletic.*

Description

Given a rooted phylogenetic tree and a set of focal tips, this function determines whether the tips form a monophyletic group.

Usage

```
is_monophyletic(tree, focal_tips, check_input=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`focal_tips`	Either an integer vector or a character vector, listing the tips to be checked for monophyly. If an integer vector, it should list tip indices (i.e. from 1 to Ntips). If a character vector, it should list tip names; in that case `tree$tip.label` must exist.
`check_input`	Logical, whether to perform basic validations of the input data. If you know for certain that your input is valid, you can set this to `FALSE` to reduce computation time.

Details

This function first finds the most recent common ancestor (MRCA) of the focal tips, and then checks if all tips descending from that MRCA fall within the focal tip set.

Value

A logical, indicating whether the focal tips form a monophyletic set.

Author(s)

Stilianos Louca

See Also

`get_mrca_of_set`

Examples

```
# generate random tree
Ntips = 100
tree  = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# pick a random subset of focal tips
focal_tips = which(sample.int(2,size=Ntips,replace=TRUE)==1)

# check if focal tips form a monophyletic group
is_monophyletic(tree, focal_tips)
```

`loglikelihood_hbd` *Galculate the log-likelihood of a homogenous birth-death model.*

Description

Given a rooted ultrametric timetree, and a homogenous birth-death (HBD) model, i.e., with speciation rate λ, extinction rate μ and sampling fraction ρ, calculate the likelihood of the tree under the model. The speciation and extinction rates may be time-dependent. “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates (in the literature this is sometimes referred to simply as “birth-death model”). Alternatively to λ and μ, the likelihood may also be calculated based on the pulled diversification rate (PDR; Louca et al. 2018) and the product $\rho \cdot \lambda(0)$, or based on the pulled speciation rate (PSR). In either case, the time-profiles of λ, μ, the PDR or the PSR are specified as piecewise polynomially functions (splines), defined on a discrete grid of ages.

Usage

```
loglikelihood_hbd(tree,
                  oldest_age      = NULL,
                  rho             = NULL,
                  rholambda0      = NULL,
                  age_grid        = NULL,
                  lambda          = NULL,
                  mu              = NULL,
                  PDR             = NULL,
                  PSR             = NULL,
                  splines_degree  = 1,
                  condition       = "stem",
                  relative_dt     = 1e-3)
```

Arguments

`tree`	A rooted ultrametric tree of class "phylo".
`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to consider when calculating the likelihood. If this is equal to or greater than the root age, then `oldest_age` is taken as the stem age, and the classical formula by Morlon et al. (2011) is used. If `oldest_age` is less than the root age, the tree is split into multiple subtrees at that age by treating every edge crossing that age as the stem of a subtree, and each subtree is considered an independent realization of the HBD model stemming at that age. This can be useful for avoiding points in the tree close to the root, where estimation uncertainty is generally higher. If `oldest_age==NULL`, it is automatically set to the root age.
`rho`	Numeric between 0 (exclusive) and 1 (inclusive), specifying the sampling fraction (ρ, aka. "rarefaction") of the tree, i.e. the fraction of extant species included. Note that if $rho < 1$, species are assumed to have been sampled randomly at equal probabilities. Can also be `NULL`, in which case `rholambda0` and `PDR` (see below) must be provided.
`rholambda0`	Strictly positive numeric, specifying the product of the sampling fraction (ρ) and the present-day speciation rate ($\lambda(0)$), units 1/time. Can be `NULL`, in which case `rarefaction`, `lambda` and `mu` must be provided.
`age_grid`	Numeric vector, listing discrete ages (time before present) on which either λ and μ, or the PDR, are specified. Listed ages must be strictly increasing, and must cover at least the full considered age interval (from 0 to `oldest_age`). Can also be `NULL` or a vector of size 1, in which case the speciation rate, extinction rate and PDR are assumed to be time-independent.
`lambda`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing speciation rates (in units 1/time) at the ages listed in `age_grid`. Speciation rates should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then either `PDR` and `rholambda0`, or `PSR` alone, must be provided.

`mu`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing extinction rates (in units 1/time)at the ages listed in `age_grid`. Extinction rates should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then `PDR` and `rholambda0`, or `PSR` alone, must be provided.
`PDR`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing pulled diversification rates (in units 1/time) at the ages listed in `age_grid`. PDRs can be negative or positive, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then either `lambda` and `mu`, or `PSR` alone, must be provided.
`PSR`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing pulled speciation rates (in units 1/time) at the ages listed in `age_grid`. PSRs should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then either `lambda` and `mu`, or `PDR` and `rholambda0`, must be provided.
`splines_degree`	Integer, either 0,1,2 or 3, specifying the polynomial degree of the provided `lambda`, `mu`, `PDR` and `PSR` (whichever applicable) between grid points in `age_grid`. For example, if `splines_degree==1`, then the provided `lambda`, `mu`, `PDR` and `PSR` are interpreted as piecewise-linear curves; if `splines_degree==2` they are interpreted as quadratic splines; if `splines_degree==3` they are interpreted as cubic splines. The `splines_degree` influences the analytical properties of the curve, e.g. `splines_degree==1` guarantees a continuous curve, `splines_degree==2` guarantees a continuous curve and continuous derivative, and so on.
`condition`	Character, either "crown", "stem" or "none" (the last one is only available if `lambda` and `mu` are given), specifying on what to condition the likelihood. If "crown", the likelihood is conditioned on the survival of the two daughter lineages branching off at the root. If "stem", the likelihood is conditioned on the survival of the stem lineage. Note that "crown" really only makes sense when `oldest_age` is equal to the root age, while "stem" is recommended if `oldest_age` differs from the root age. "none" is usually not recommended and is only available when `lambda` and `mu` are provided.
`relative_dt`	Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

Details

This function supports two alternative parameterizations of HBD models, either using the speciation and extinction rates and sampling fraction (λ, μ and ρ), or using the pulled diversification rate (PDR) and the product $\rho \cdot \lambda(0)$ (sampling fraction times present-day speciation rate), or using the pulled speciation rate (PSR). The latter two options should be interpreted as a parameterization of congruence classes, i.e. sets of models that have the same likelihood, rather than specific models, since multiple combinations of λ, μ and ρ can have identical PDRs, $\rho \cdot \lambda(0)$ and PSRs (Louca and Pennell, in review).

For large trees the asymptotic time complexity of this function is O(Nips). The tree may include monofurcations as well as multifurcations, and the likelihood formula accounts for those (i.e., as if monofurcations were omitted and multifurcations were expanded into bifurcations).

Value

A named list with the following elements:

`success`	Logical, indicating whether the calculation was successful. If `FALSE`, then the returned list includes an additional '`error`' element (character) containing a description of the error; all other return variables may be undefined.
`loglikelihood`	Numeric. If `success==TRUE`, this will be the natural logarithm of the likelihood of the tree under the given model.

Author(s)

Stilianos Louca

References

H. Morlon, T. L. Parsons, J. B. Plotkin (2011). Reconciling molecular phylogenies with the fossil record. Proceedings of the National Academy of Sciences. 108:16327-16332.

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

S. Louca and M. W. Pennell (in review as of 2019)

See Also

`simulate_deterministic_hbd`

`fit_hbd_model_parametric`

`fit_hbd_model_on_grid`

`fit_hbd_pdr_on_grid`

`fit_hbd_pdr_parametric`

Examples

```
# generate a random tree with constant rates
Ntips  = 100
params = list(birth_rate_factor=1, death_rate_factor=0.2, rarefaction=0.5)
tree   = generate_random_tree(params, max_tips=Ntips, coalescent=TRUE)$tree

# get the loglikelihood for an HBD model with the same parameters that generated the tree
# in particular, assuming time-independent speciation & extinction rates
LL = loglikelihood_hbd( tree,
                        rho       = params$rarefaction,
                        age_grid  = NULL, # assume time-independent rates
                        lambda    = params$birth_rate_factor,
                        mu        = params$death_rate_factor)
```

```
if(LL$success){
  cat(sprintf("Loglikelihood for constant-rates model = %g\n",LL$loglikelihood))
}

# get the likelihood for a model with exponentially decreasing (in forward time) lambda & mu
beta      = 0.01 # exponential decay rate of lambda over time
age_grid  = seq(from=0, to=100, by=0.1) # choose a sufficiently fine age grid
lambda    = 1*exp(beta*age_grid) # define lambda on the age grid
mu        = 0.2*lambda # assume similarly shaped but smaller mu
LL = loglikelihood_hbd( tree,
                        rho       = params$rarefaction,
                        age_grid  = age_grid,
                        lambda    = lambda,
                        mu        = mu)
if(LL$success){
  cat(sprintf("Loglikelihood for exponential-rates model = %g\n",LL$loglikelihood))
}
```

map_to_state_space *Map states of a discrete trait to integers.*

Description

Given a list of states (e.g., for each tip in a tree), map the unique states to integers 1,..,Nstates, where Nstates is the number of possible states. This function can be used to translate states that are originally represented by characters or factors, into integer states as required by ancestral state reconstruction and hidden state prediction functions in this package.

Usage

```
map_to_state_space(raw_states, fill_gaps=FALSE,
                   sort_order="natural", include_state_values=FALSE)
```

Arguments

`raw_states`	A vector of values (states), each of which can be converted to a different character. This list can include the same value multiple times, for example if values represent the trait's states for tips in a tree.
`fill_gaps`	Logical. If `TRUE`, then states are converted to integers using `as.integer(as.character())`, and then all missing intermediate integer values are included as additional possible states. For example, if `raw_states` contained the values 2,4,6, then 3 and 5 are assumed to also be possible states.
`sort_order`	Character, specifying the order in which raw_states should be mapped to ascending integers. Either "natural" or "alphabetical". If "natural", numerical parts of characters are sorted numerically, e.g. as in "3"<"a2"<"a12"<"b1".
`include_state_values`	Logical, specifying whether to also return a numerical version of the unique states. For example, the states "3","a2","4.5" will be mapped to the numeric values 3, NA, 4.5.

Details

Several ancestral state reconstruction and hidden state prediction algorithms in the `castor` package (e.g., `asr_max_parsimony`) require that the focal trait's states are represented by integer indices within 1,..,Nstates. These indices are then associated, afor example, with column and row indices in the transition cost matrix (in the case of maximum parsimony reconstruction) or with column indices in the returned matrix containing marginal ancestral state probabilities (e.g., in `asr_mk_model`). The function `map_to_state_space` can be used to conveniently convert a set of discrete states into integers, for use with the aforementioned algorithms.

Value

A list with the following elements:

`Nstates`	Integer. Number of possible states for the trait, based on the unique values encountered in `raw_states` (after conversion to characters). This may be larger than the number of unique values in `raw_states`, if `fill_gaps` was set to `TRUE`.
`state_names`	Character vector of size Nstates, storing the original name (character version) of each state. For example, if `raw_states` was `c("b1","3","a12","a2","b1","a2")` and `sort_order=="natural"`, then `Nstates` will be 4 and `state_names` will be `c("3","a2","a12","b1")`.
`state_values`	Optional, only included if `include_state_values==TRUE`. A numeric vector of size `Nstates`, providing the numerical value for each unique state.
`mapped_states`	Integer vector of size equal to `length(raw_states)`, listing the integer representation of each value in `raw_states`.
`name2index`	An integer vector of size Nstates, with `names(name2index)` set to `state_names`. This vector can be used to map any new list of states (in character format) to their integer representation. In particular, `name2index[as.character(raw_states)]` is equal to `mapped_states`.

Author(s)

Stilianos Louca

Examples

```
# generate a sequence of random states
unique_states = c("b","c","a")
raw_states = unique_states[sample.int(3,size=10,replace=TRUE)]

# map to integer state space
mapping = map_to_state_space(raw_states)

cat(sprintf("Checking that original unique states is the same as the one inferred:\n"))
print(unique_states)
print(mapping$state_names)

cat(sprintf("Checking reversibility of mapping:\n"))
```

```
print(raw_states)
print(mapping$state_names[mapping$mapped_states])
```

---

merge_short_edges *Eliminate short edges in a tree by merging nodes into multifurcations.*

Description

Given a rooted phylogenetic tree and an edge length threshold, merge nodes/tips into multifurcations when their incoming edges are shorter than the threshold.

Usage

```
merge_short_edges(tree,
                  edge_length_epsilon = 0,
                  force_keep_tips     = TRUE,
                  new_tip_prefix      = "ex.node.tip.")
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`edge_length_epsilon`	Non-negative numeric, specifying the maximum edge length for an edge to be considered “short” and thus to be eliminated. Typically 0 or some small positive number.
`force_keep_tips`	Logical. If `TRUE`, then tips are always kept, even if their incoming edges are shorter than `edge_length_epsilon`. If `FALSE`, then tips with short incoming edges are removed from the tree; in that case some nodes may become tips.
`new_tip_prefix`	Character or `NULL`, specifying the prefix to use for new tip labels stemming from nodes. Only relevant if `force_keep_tips==FALSE`. If `NULL`, then labels of tips stemming from nodes will be the node labels from the original tree (in this case the original tree should include node labels).

Details

The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). Whenever a short edge is eliminated, the edges originating from its child are elongated according to the short edge's length. The corresponding grand-children become children of the short edge's parent. Short edges are eliminated in a depth-first-search manner, i.e. traversing from the root to the tips.

Note that existing monofurcations are retained. If `force_keep_tips==FALSE`, then new monofurcations may also be introduced due to tips being removed.

This function is conceptually similar to the function `ape::di2multi`.

Value

A list with the following elements:

`tree`	A new rooted tree of class "phylo", containing the (potentially multifurcating) tree.
`new2old_clade`	Integer vector of length equal to the number of tips+nodes in the new tree, with values in 1,..,Ntips+Nnodes, mapping tip/node indices of the new tree to tip/node indices in the original tree.
`new2old_edge`	Integer vector of length equal to the number of edges in the new tree, with values in 1,..,Nedges, mapping edge indices of the new tree to edge indices in the original tree.
`Nedges_removed`	Integer. Number of edges that have been eliminated.

Author(s)

Stilianos Louca

See Also

`multifurcations_to_bifurcations`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_factor=1),max_tips=Ntips)$tree

# set some edge lengths to zero
tree$edge.length[sample.int(n=Ntips, size=10, replace=FALSE)] = 0

# print number of edges
cat(sprintf("Original tree has %d edges\n",nrow(tree$edge)))

# eliminate any edges of length zero
merged = merge_short_edges(tree, edge_length_epsilon=0)$tree

# print number of edges
cat(sprintf("New tree has %d edges\n",nrow(merged$edge)))
```

`multifurcations_to_bifurcations` *Expand multifurcations to bifurcations.*

Description

Eliminate multifurcations from a phylogenetic tree, by replacing each multifurcation with multiple bifurcations.

Usage

```
multifurcations_to_bifurcations(tree, dummy_edge_length=0,
                                new_node_basename="node.",
                                new_node_start_index=NULL)
```

Arguments

`tree`	A tree of class "phylo".
`dummy_edge_length`	Non-negative numeric. Length to be used for new (dummy) edges when breaking multifurcations into bifurcations. Typically this will be 0, but can also be a positive number if zero edge lengths are not desired in the returned tree.
`new_node_basename`	Character. Name prefix to be used for added nodes (e.g. "node." or "new.node."). Only relevant if the input tree included node labels.
`new_node_start_index`	Integer. First index for naming added nodes. Can also be `NULL`, in which case this is set to Nnodes+1, where Nnodes is the number of nodes in the input tree.

Details

For each multifurcating node (i.e. with more than 2 children), all children but one will be placed on new bifurcating nodes, connected to the original node through one or more dummy edges.

The input tree need not be rooted, however descendance from each node is inferred based on the direction of edges in `tree$edge`. The input tree may include multifurcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child). Monofurcations are kept in the returned tree.

All tips and nodes in the input tree retain their original indices, however the returned tree may include additional nodes and edges. Edge indices may change.

If `tree$edge.length` is missing, then all edges in the input tree are assumed to have length 1. The returned tree will include `edge.length`, with all new edges having length equal to `dummy_edge_length`.

Value

A list with the following elements:

`tree`	A new tree of class "phylo", containing only bifurcations (and monofurcations, if these existed in the input tree).
`old2new_edge`	Integer vector of length Nedges, mapping edge indices in the old tree to edge indices in the new tree.
`Nnodes_added`	Integer. Number of nodes added to the new tree.

Author(s)

Stilianos Louca

See Also

`collapse_monofurcations`

Examples

```
# generate a random multifurcating tree
Ntips = 1000
tree = generate_random_tree(list(birth_rate_intercept=1), Ntips, Nsplits=5)$tree

# expand multifurcations to bifurcations
new_tree = multifurcations_to_bifurcations(tree)$tree

# print summary of old and new tree
cat(sprintf("Old tree has %d nodes\n",tree$Nnode))
cat(sprintf("New tree has %d nodes\n",new_tree$Nnode))
```

`pick_random_tips` *Pick random subsets of tips on a tree.*

Description

Given a rooted phylogenetic tree, this function picks random subsets of tips by traversing the tree from root to tips, choosing a random child at each node until reaching a tip. Multiple random independent subsets can be generated if needed.

Usage

```
pick_random_tips( tree,
                  size             = 1,
                  Nsubsets         = 1,
                  with_replacement = TRUE,
                  drop_dims        = TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`size`	Integer. The size of each random subset of tips.
`Nsubsets`	Integer. Number of independent subsets to pick.
`with_replacement`	Logical. If `TRUE`, each tip can be picked multiple times within a subset (i.e. are "replaced" in the urn). If `FALSE`, tips are picked without replacement in each subset. In that case, `size` must not be greater than the number of tips in the tree.
`drop_dims`	Logical, specifying whether to return a vector (instead of a matrix) if `Nsubsets==1`.

Details

If `with_replacement==TRUE`, then each child of a node is equally probable to be traversed and each tip can be included multiple times in a subset. If `with_replacement==FALSE`, then only children with at least one descending tip not included in the subset remain available for traversal; each available child of a node has equal probability to be traversed. In any case, it is always possible for separate subsets to include the same tips.

This random sampling algorithm differs from a uniform sampling of tips at equal probabilities; instead, this algorithm ensures that sister clades have equal probabilities to be picked (if `with_replacement==TRUE` or if `size`«Ntips).

The time required by this function per random subset decreases with the number of subsets requested.

Value

A 2D integer matrix of size Nsubsets x size, with each row containing indices of randomly picked tips (i.e. in 1,..,Ntips) within a specific subset. If `drop_dims==TRUE` and `Nsubsets==1`, then a vector is returned instead of a matrix.

Author(s)

Stilianos Louca

Examples

```
# generate random tree
Ntips = 1000
tree  = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# pick random tip subsets
Nsubsets = 100
size     = 50
subsets = pick_random_tips(tree, size, Nsubsets, with_replacement=FALSE)

# count the number of times each tip was picked in a subset ("popularity")
popularities = table(subsets)

# plot histogram of tip popularities
hist(popularities,breaks=20,xlab="popularity",ylab="# tips",main="tip popularities")
```

`read_tree` *Load a tree from a string or file in Newick (parenthetic) format.*

Description

Load a phylogenetic tree from a file or a string, in Newick (parenthetic) format. Any valid Newick format is acceptable.

Usage

```
read_tree( string  = "",
           file    = "",
           edge_order              = "cladewise",
           include_edge_lengths    = TRUE,
           include_node_labels     = TRUE,
           underscores_as_blanks   = FALSE,
           check_label_uniqueness  = FALSE)
```

Arguments

`string`	A character containing a single tree in Newick format. Can be used alternatively to `file`.
`file`	Character, a path to an input text file containing a single tree in Newick format. Can be used alternatively to `string`.
`edge_order`	Character, one of "cladewise" or "pruningwise", specifying the order in which edges should be listed in the returned tree. This does not influence the topology of the tree or the tip/node labeling, it only affects the way edges are numbered internally.
`include_edge_lengths`	Logical, specifying whether edge lengths (if available) should be included in the generated tree.
`include_node_labels`	Logical, specifying whether node labels (if available) should be included in the generated tree.
`underscores_as_blanks`	Logical, specifying whether underscores ("_") in tip and node labels should be replaced by spaces (" "). This is common behavior in other tree parsers. In any case, tip and node labels are also allowed to contain explicit whitespace (other than newlines).
`check_label_uniqueness`	Logical, specifying whether to check if all tip labels are unique.

Details

This function is comparable to (but typically much faster than) the `ape` function `read.tree`. The function supports trees with monofurcations and multifurcations, trees with or without tip/node labels, and trees with or without edge lengths. The time complexity is linear in the number of edges in the tree.

Either `file` or `string` must be specified, but not both. The tree may be arbitrarily split across multiple lines, but no other non-whitespace text is permitted in `string` or in the input file. Flanking whitespace (space, tab, newlines) is ignored.

Value

A single rooted phylogenetic tree in "phylo" format.

Author(s)

Stilianos Louca

See Also

`write_tree`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=100)$tree

# obtain a string representation of the tree in Newick format
Newick_string = write_tree(tree)

# re-parse tree from string
parsed_tree = read_tree(Newick_string)
```

`reconstruct_past_diversification` *Reconstruct past diversification dynamics from a diversity time series.*

Description

Given a time series of past diversities (coalescent or not), this function estimates instantaneous birth (speciation) and death (extinction) rates that would lead to the observed diversity time series. The function is based on a deterministic model (or the continuum limit of a stochastic cladogenic model), in which instantaneous birth and death rates lead to a predictable growth of a tree (one new species per birth event). The reconstruction is non-parametric, i.e. does not rely on fitting a parameterized model. The reconstruction is only accurate in the deterministic limit, i.e. for high diversities where the stochastic nature of the cladogenic process diminishes. Of particular importance is the case where the time series is coalescent, i.e. represents the diversity (lineages-through-time) that would be represented in a coalescent tree with extinctions.

Usage

```
reconstruct_past_diversification( times,
                                  diversities,
                                  birth_rates_pc             = NULL,
                                  rarefaction                = NULL,
                                  discovery_fractions        = NULL,
                                  discovery_fraction_slopes = NULL,
                                  max_age                    = NULL,
                                  coalescent                 = FALSE,
                                  smoothing_span             = 0,
                                  smoothing_order            = 1)
```

Arguments

`times`	Numeric vector, listing the times at which diversities are given. Values must be in ascending order.
`diversities`	Numeric vector of the same size as `times`, listing diversities (coalescent or not) at each time point.
`birth_rates_pc`	Numeric vector of the same size as `times`, listing known or assumed per-capita birth rates (speciation rates). Can also be of size 1, in which case the same per-capita birth rate is assumed throughout. Alternatively if `coalescent==TRUE`, then this vector can also be empty, in which case a constant per-capita birth rate is assumed and estimated from the slope of the coalescent diversities at the last time point. The last alternative is not available when `coalescent==FALSE`.
`rarefaction`	Numeric between 0 and 1. Optional rarefaction fraction assumed for the diversities at the very end. Set to 1 to assume no rarefaction was performed.
`discovery_fractions`	Numeric array of size Ntimes, listing the fractions of extant lineages represented in the tree over time. Hence, `discovery_fraction[t]` is the probability that a lineage at time `times[t]` with extant representatives will be represented in the tree. Can be used as an alternative to `rarefaction`, for example if discovery of extant species is non-random or phylogenetically biased. Experimental, so leave this `NULL` if you don't know what it means.
`discovery_fraction_slopes`	Numeric array of size Ntimes, listing the 1st derivative of `discovery_fractions` (w.r.t. time) over time. If `NULL`, this will be estimated from `discovery_fractions` via basic finite differences if needed. Experimental, so leave this `NULL` if you don't know what it means.
`max_age`	Numeric. Optional maximum distance from the end time to be considered. If `NULL` or <=0 or `Inf`, all provided time points are considered.
`coalescent`	Logical, indicating whether the provided diversities are from a coalescent tree (only including clades with extant representatives) or total diversities (extant species at each time point).
`smoothing_span`	Non-negative integer. Optional sliding window size (number of time points) for smoothening the diversities time series via Savitzky-Golay-filter. If <=2, no smoothing is done. Smoothening the time series can reduce the effects of noise on the reconstructed diversity dynamics.
`smoothing_order`	Integer between 1 and 4. Polynomial order of the Savitzky-Golay smoothing filter to be applied. Only relevant if `smoothing_span>2`. A value of 1 or 2 is typically recommended.

Details

For a coalescent diversity time series $N_c(\tau)$ at various ages τ (distance from the end of the time series), reconstruction of the total diversity $N(\tau)$ is based on the following formulas:

$$E(\tau) = 1 + \frac{\nu(\tau)}{\beta(\tau)},$$

$$N(\tau) = \frac{N_c}{1 - E(\tau)},$$

$$\nu(\tau) = \frac{1}{N_c(\tau)} \frac{dN_c(\tau)}{d\tau}$$

where $E(\tau)$ is the probability that a clade of size 1 at age τ went extinct by the end of the time series and β is the per-capita birth rate. If the per-capita birth rate is not explicitly provided for each time point (see argument `birth_rate_pc`), the function assumes that the per-capita birth rate (speciation rate) is constant at all times. If `birth_rates_pc==NULL` and `coalescent==TRUE`, the constant speciation rate is estimated as

$$\beta = -\frac{\nu(0)}{\rho},$$

where ρ is the fraction of species kept after rarefaction (see argument `rarefaction`).

Assuming a constant speciation rate may or may not result in accurate estimates of past total diversities and other quantities. If a time-varying speciation rate is suspected but not known, additional information on past diversification dynamics may be obtained using modified ("pulled") quantities that partly resemble the classical extinction rate, diversification rate and total diversity. Such quantities are the "pulled diversification rate":

$$\eta(\tau) = \delta(\tau) - \beta(\tau) + \frac{1}{\beta(\tau)} \frac{d\beta}{d\tau},$$

the "pulled extinction rate":

$$\delta_p(\tau) = \delta(\tau) + (\beta_o - \beta(\tau)) - \frac{1}{\beta(\tau)} \frac{d\beta}{d\tau},$$

and the "pulled total diversity":

$$N_p(\tau) = N(\tau) \cdot \frac{\beta_o}{\beta(\tau)},$$

where β_o is the provided or estimated (if not provided) speciation rate at the last time point. The advantage of these quantities is that they can be estimated from the coalescent diversities (lineages-through-time) without any assumptions on how β and δ varied over time. The disadvantage is that they differ from their "non-pulled" quantities ($\beta - \delta$, δ and N), in cases where β varied over time.

Value

A named list with the following elements:

`success`	Logical, specifying whether the reconstruction was successful. If `FALSE`, the remaining elements may not be defined.
`Ntimes`	Integer. Number of time points for which reconstruction is returned.
`total_diversities`	Numeric vector of the same size as `times`, listing the total diversity at each time point (number of extant lineages at each time point). If `coalescent==FALSE`, then these are the same as the `diversities` passed to the function.

`coalescent_diversities`
: Numeric vector of the same size as `times`, listing the coalescent diversities at each time point (number of species with at least one extant descendant at the last time point). If `coalescent==TRUE`, then these are the same as the `diversities` passed to the function.

`birth_rates`
: Numeric vector of the same size as `times`, listing the estimated birth rates (speciation events per time unit).

`death_rates`
: Numeric vector of the same size as `times`, listing the estimated death rates (extinction events per time unit).

`Psurvival`
: Numeric vector of the same size as `times`, listing the estimated fraction of lineages at each time point that eventually survive. `Psurvival[i]` is the probability that a clade of size 1 at time `times[i]` will be extant by the end of the time series. May be `NULL` in some cases.

`Pdiscovery`
: Numeric vector of the same size as `times`, listing the estimated fraction of lineages at each time point that are eventually discovered, provided that they survive. `Pdiscovery[i]` is the probability that a clade of size 1 at time `times[i]` that is extant by the end of the time series, will be discovered. May be `NULL` in some cases.

`Prepresentation`
: Numeric vector of the same size as `times`, listing the estimated fraction of lineages at each time point that eventually survive and are discovered. `Prepresentation[i]` is the probability that a clade of size 1 at time `times[i]` will be extant by the end of the time series and visible in the coalescent tree after rarefaction. Note that Prepresentation = Psurvival * Pdiscovery. May be `NULL` in some cases.

`total_births`
: Numeric, giving the estimated total number of birth events that occurred between times `T-max_age` and `T`, where `T` is the last time point of the time series.

`total_deaths`
: Numeric, giving the estimated total number of death events that occurred between times `T-max_age` and `T`, where `T` is the last time point of the time series.

`last_birth_rate_pc`
: The provided or estimated (if not provided) speciation rate at the last time point. This corresponds to the birth rate divided by the estimated true diversity (prior to rarefaction) at the last time point.

`last_death_rate_pc`
: The estimated extinction rate at the last time point. This corresponds to the death rate divided by the estimated true diversity (prior to rarefaction) at the last time point.

`pulled_diversification_rates`
: Numeric vector of the same size as `times`, listing the estimated pulled diversification rates.

`pulled_extinction_rates`
: Numeric vector of the same size as `times`, listing the estimated pulled extinction rates.

`pulled_total_diversities`
: Numeric vector of the same size as `times`, listing the estimated pulled total diversities.

Author(s)

Stilianos Louca

See Also

`generate_random_tree`, `fit_tree_model` `count_lineages_through_time`

Examples

```
##################################################
# EXAMPLE 1

# Generate a coalescent tree
params = list(birth_rate_intercept  = 0,
              birth_rate_factor     = 1,
              birth_rate_exponent   = 1,
              death_rate_intercept  = 0,
              death_rate_factor     = 0.05,
              death_rate_exponent   = 1.3,
              rarefaction           = 1)
simulation = generate_random_tree(params,max_time_eq=1,coalescent=TRUE)
tree = simulation$tree
time_span = simulation$final_time - simulation$root_time
cat(sprintf("Generated tree has %d tips, spans %g time units\n",length(tree$tip.label),time_

# Calculate diversity time series from the tree
counter = count_lineages_through_time(tree, times=seq(0,0.99*time_span,length.out=100))

# print coalescent diversities
print(counter$lineages)

# reconstruct diversification dynamics based on diversity time series
results = reconstruct_past_diversification( counter$times,
                                            counter$lineages,
                                            coalescent      = TRUE,
                                            smoothing_span  = 3,
                                            smoothing_order = 1)

# print reconstructed total diversities
print(results$total_diversities)

# plot coalescent and reconstructed true diversities
matplot(x     = counter$times,
        y     = matrix(c(counter$lineages,results$total_diversities), ncol=2, byrow=FALSE),
        type  = "b",
        xlab  = "time",
        ylab  = "# clades",
        lty   = c(1,2), pch = c(1,0), col = c("red","blue"))
legend( "topleft",
        legend  = c("coalescent (simulated)","true (reconstructed)"),
        col     = c("red","blue"), lty = c(1,2), pch = c(1,0));
```

```
#####################################################
# EXAMPLE 2

# Generate a non-coalescent tree
params = list(birth_rate_intercept  = 0,
              birth_rate_factor     = 1,
              birth_rate_exponent   = 1,
              death_rate_intercept  = 0,
              death_rate_factor     = 0.05,
              death_rate_exponent   = 1.3,
              rarefaction           = 1)
simulation = generate_random_tree(params,max_time_eq=1,coalescent=FALSE)
tree = simulation$tree
time_span = simulation$final_time - simulation$root_time
cat(sprintf("Generated tree has %d tips, spans %g time units\n",length(tree$tip.label),time_

# Calculate diversity time series from the tree
counter = count_lineages_through_time(tree, times=seq(0,0.99*time_span,length.out=100))

# print true diversities
print(counter$lineages)

# reconstruct diversification dynamics based on diversity time series
results = reconstruct_past_diversification( counter$times,
                                            counter$lineages,
                                            birth_rates_pc  = params$birth_rate_factor,
                                            coalescent      = FALSE,
                                            smoothing_span  = 3,
                                            smoothing_order = 1)

# print coalescent diversities
print(results$coalescent_diversities)

# plot coalescent and reconstructed true diversities
matplot(x     = counter$times,
        y     = matrix(c(results$coalescent_diversities,counter$lineages), ncol=2, byrow=FAL
        type  = "b",
        xlab  = "time",
        ylab  = "# clades",
        lty   = c(1,2), pch = c(1,0), col = c("red","blue"))
legend( "topleft",
        legend  = c("coalescent (reconstructed)","true (simulated)"),
        col     = c("red","blue"), lty = c(1,2), pch = c(1,0));
```

`reorder_tree_edges` *Reorder tree edges in preorder or postorder.*

Description

Given a rooted tree, this function reorders the rows in `tree$edge` so that they are listed in preorder (root–>tips) or postorder (tips–>root) traversal.

Usage

```
reorder_tree_edges(tree, root_to_tips=TRUE,
                   depth_first_search=TRUE,
                   index_only=FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`root_to_tips`	Logical, specifying whether to sort edges in preorder traversal (root–>tips), rather than in postorder traversal (tips–>roots).
`depth_first_search`	Logical, specifying whether the traversal (or the reversed traversal, if `root_to_tips` is `FALSE`) should be in depth-first-search format rather than breadth-first-search format.
`index_only`	Whether the function should only return a vector listing the reordered row indices of the edge matrix, rather than a modified tree.

Details

This function does not change the tree structure, nor does it affect tip/node indices and names. It merely changes the order in which edges are listed in the matrix `tree$edge`, so that edges are listed in preorder or postorder traversal. Preorder traversal guarantees that each edge is listed before any of its descending edges. Likewise, postorder guarantees that each edge is listed after any of its descending edges.

With options `root_to_tips=TRUE` and `depth_first_search=TRUE`, this function is analogous to the function `reorder` in the `ape` package with option `order="cladewise"`.

The tree can include multifurcations (nodes with more than 2 children) as well as monofurcations (nodes with 1 child). This function has asymptotic time complexity O(Nedges).

Value

If `index_only==FALSE`, a tree object of class "phylo", with the rows in `edge` reordered such that they are listed in direction root–>tips (if `root_to_tips==TRUE`) or tips–>root. The vector `tree$edge.length` will also be updated in correspondence. Tip and node indices and names remain unchanged.

If `index_only=TRUE`, an integer vector (X) of size Nedges, listing the reordered row indices of `tree$edge`, i.e. such that `tree$edge[X,]` would be the reordered edge matrix.

Author(s)

Stilianos Louca

See Also

get_tree_traversal_root_to_tips

Examples

```
## Not run:
postorder_tree = reorder_tree_edges(tree, root_to_tips=FALSE)

## End(Not run)
```

---

root_at_midpoint *Root or re-root a tree at the midpoint node.*

Description

Given a tree (rooted or unrooted), this function changes the direction of edges (`tree$edge`) such that the midpoint node becomes the new root (i.e. has no incoming edges and all other tips and nodes descend from it). The number of tips and the number of nodes remain unchanged. The midpoint node is the node whose maximum distance to any tip is smallest.

Usage

```
root_at_midpoint( tree,
                  update_indices  = TRUE,
                  as_edge_counts  = FALSE,
                  is_rooted       = FALSE)
```

Arguments

`tree`	A tree object of class "phylo". Can be unrooted or rooted (but see option `is_rooted`).
`update_indices`	Logical, specifying whether to update the node indices such that the new root is the first node in the list (as is common convention). This will modify `tree$node.label` (if it exists) and also the node indices listed in `tree$edge`.
`as_edge_counts`	Logical, specifying whether phylogenetic distances should be measured as cumulative edge counts. This is the same if all edges had length 1.
`is_rooted`	Logical, specifying whether the input tree can be assumed to be rooted. If you are not certain that the tree is rooted, set this to `FALSE`.

Details

The input tree may include an arbitrary number of incoming and outgoing edges per node (but only one edge per tip), and the direction of these edges can be arbitrary. Of course, the undirected graph defined by all edges must still be a valid tree. Only set `is_rooted=TRUE` if you are sure that the input tree is rooted.

If `update_indices==FALSE`, then node indices remain unchanged. If `update_indices==TRUE` (default), then node indices are modified such that the new root is the first node (i.e. with index Ntips+1 in `edge` and with index 1 in `node.label`), as is common convention. Setting `update_indices=FALSE` reduces the computation required for rerooting. Tip indices always remain unchanged.

The asymptotic time complexity of this function is O(Nedges).

Value

A tree object of class "phylo", with the `edge` element modified such that the maximum distance of the root to any tip is minimized. The elements `tip.label`, `edge.length` and `root.edge` (if they exist) are the same as for the input tree. If `update_indices==FALSE`, then the element `node.label` will also remain the same.

Author(s)

Stilianos Louca

See Also

`root_via_outgroup`, `root_at_node`, `root_in_edge`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# reroot the tree at its midpoint node
tree = root_at_midpoint(tree)
```

`root_at_node` *Root or re-root a tree at a specific node.*

Description

Given a tree (rooted or unrooted) and a specific node, this function changes the direction of edges (`tree$edge`) such that the designated node becomes the root (i.e. has no incoming edges and all other tips and nodes descend from it). The number of tips and the number of nodes remain unchanged.

Usage

```
root_at_node(tree, new_root_node, update_indices=TRUE)
```

Arguments

`tree`	A tree object of class "phylo". Can be unrooted or rooted.
`new_root_node`	Character or integer specifying the name or index, respectively, of the node to be turned into root. If an integer, it must be between 1 and `tree$Nnode`. If a character, it must be a valid entry in `tree$node.label`.
`update_indices`	Logical, specifying whether to update the node indices such that the new root is the first node in the list (as is common convention). This will modify `tree$node.label` (if it exists) and also the node indices listed in `tree$edge`.

Details

The input tree may include an arbitrary number of incoming and outgoing edges per node (but only one edge per tip), and the direction of these edges can be arbitrary. Of course, the undirected graph defined by all edges must still be a valid tree. The asymptotic time complexity of this function is O(Nedges).

If `update_indices==FALSE`, then node indices remain unchanged. If `update_indices==TRUE` (default), then node indices are modified such that the new root is the first node (i.e. with index Ntips+1 in `edge` and with index 1 in `node.label`). This is common convention, but it may be undesirable if, for example, you are looping through all nodes in the tree and are only temporarily designating them as root. Setting `update_indices=FALSE` also reduces the computation required for rerooting. Tip indices always remain unchanged.

Value

A tree object of class "phylo", with the `edge` element modified such that the node `new_root_node` is root. The elements `tip.label`, `edge.length` and `root.edge` (if they exist) are the same as for the input tree. If `update_indices==FALSE`, then the element `node.label` will also remain the same.

Author(s)

Stilianos Louca

See Also

`root_via_outgroup`, `root_at_midpoint`, `root_in_edge`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree
```

```
# reroot the tree at the 20-th node
new_root_node = 20
tree = root_at_node(tree, new_root_node, update_indices=FALSE)

# find new root index and compare with expectation
cat(sprintf("New root is %d, expected at %d\n",find_root(tree),new_root_node+Ntips))
```

---

root_in_edge — *Root or re-root a tree in the middle of an edge.*

Description

Given a tree (rooted or unrooted), this function places the new root in the middle of a specified edge, effectively adding one more node, one more edge and changing the direction of edges as required.

Usage

```
root_in_edge( tree,
              root_edge,
              new_root_name = "root",
              collapse_monofurcations = TRUE)
```

Arguments

`tree`	A tree object of class "phylo". Can be unrooted or rooted.
`root_edge`	Integer, index of the edge into which the new root is to be placed. Must be between 1 and Nedges.
`new_root_name`	Character, specifying the node name to use for the new root. Only used if `tree$node.label` is not `NULL`.
`collapse_monofurcations`	Logical, specifying whether monofurcations in the rerooted tree (e.g. stemming from the old root) should be collapsed by connecting incoming edges with outgoing edges.

Details

The input tree may include an arbitrary number of incoming and outgoing edges per node (but only one edge per tip), and the direction of these edges can be arbitrary. Of course, the undirected graph defined by all edges must still be a valid tree.

The number of tips in the rerooted tree remains unchanged, the number of nodes is increased by 1. Node indices may be modified. Tip indices always remain unchanged.

The asymptotic time complexity of this function is O(Nedges).

Value

A tree object of class "phylo", representing the (re-)rooted phylogenetic tree. The element `tip.label` is the same as for the input tree, but all other elements may have changed.

Author(s)

Stilianos Louca

See Also

`root_via_outgroup`, `root_at_node`, `root_at_midpoint`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# reroot the tree inside some arbitrary edge
focal_edge = 120
tree = root_in_edge(tree, focal_edge)
```

`root_via_outgroup` *Root or re-root a tree based on an outgroup tip.*

Description

Given a tree (rooted or unrooted) and a specific tip ("outgroup"), this function changes the direction of edges (`tree$edge`) such that the outgroup's parent node becomes the root. The number of tips and the number of nodes remain unchanged.

Usage

```
root_via_outgroup(tree, outgroup, update_indices=TRUE)
```

Arguments

`tree`	A tree object of class "phylo". Can be unrooted or rooted.
`outgroup`	Character or integer specifying the name or index, respectively, of the outgroup tip. If an integer, it must be between 1 and Ntips. If a character, it must be a valid entry in `tree$tip.label`.
`update_indices`	Logical, specifying whether to update the node indices such that the new root is the first node in the list (as is common convention). This will modify `tree$node.label` (if it exists) and also the node indices listed in `tree$edge`.

Details

The input tree may include an arbitrary number of incoming and outgoing edges per node (but only one edge per tip), and the direction of these edges can be arbitrary. Of course, the undirected graph defined by all edges must still be a valid tree. The asymptotic time complexity of this function is O(Nedges).

If `update_indices==FALSE`, then node indices remain unchanged. If `update_indices==TRUE` (default), then node indices are modified such that the new root is the first node (i.e. with index Ntips+1 in `edge` and with index 1 in `node.label`). This is common convention, but it may be undesirable in some cases. Setting `update_indices=FALSE` also reduces the computation required for rerooting. Tip indices always remain unchanged.

Value

A tree object of class "phylo", with the `edge` element modified such that the outgroup tip's parent node is root. The elements `tip.label`, `edge.length` and `root.edge` (if they exist) are the same as for the input tree. If `update_indices==FALSE`, then the element `node.label` will also remain the same.

Author(s)

Stilianos Louca

See Also

`root_at_node`, `root_at_midpoint`, `root_in_edge`

Examples

```
# generate a random tree
Ntips = 100
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=Ntips)$tree

# reroot the tree using the 1st tip as outgroup
outgroup = 1
tree = root_via_outgroup(tree, outgroup, update_indices=FALSE)

# find new root index
cat(sprintf("New root is %d\n",find_root(tree)))
```

`simulate_bm_model` *Simulate a Brownian motion model for multivariate trait co-evolution.*

Description

Given a rooted phylogenetic tree and a Brownian motion (BM) model for the co-evolution of one or more continuous (numeric) unbounded traits, simulate random outcomes of the model on all nodes and/or tips of the tree. The function traverses nodes from root to tips and randomly assigns a multivariate state to each node or tip based on its parent's previously assigned state and the specified model parameters. The generated states have joint distributions consistent with the multivariate BM model. Optionally, multiple independent simulations can be performed using the same model.

Usage

```
simulate_bm_model(tree, diffusivity=NULL, sigma=NULL,
                  include_tips=TRUE, include_nodes=TRUE,
                  root_states=NULL, Nsimulations=1, drop_dims=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`diffusivity`	Either NULL, or a single number, or a 2D quadratic positive definite symmetric matrix of size Ntraits x Ntraits. Diffusivity matrix ("D") of the multivariate Brownian motion model (in units trait^2/edge_length). The convention is that if the root's state is fixed, then the covariance matrix of a node's state at distance L from the root will be $2LD$ (see mathematical details below).
`sigma`	Either NULL, or a single number, or a 2D matrix of size Ntraits x Ndegrees, where Ndegrees refers to the degrees of freedom of the model. Noise-amplitude coefficients of the multivariate Brownian motion model (in units trait/sqrt(edge_length)). This can be used as an alternative way to specify the Brownian motion model instead of through the diffusivity D. Note that $sigma \cdot \sigma^T = 2D$ (see mathematical details below).
`include_tips`	Include random states for the tips. If `FALSE`, no states will be returned for tips.
`include_nodes`	Include random states for the nodes. If `FALSE`, no states will be returned for nodes.
`root_states`	Numeric matrix of size NR x Ntraits (where NR can be arbitrary), specifying the state of the root for each simulation. If NR is smaller than `Nsimulations`, values in `root_states` are recycled in rotation. If `root_states` is `NULL` or empty, then the root state is set to 0 for all traits in all simulations.
`Nsimulations`	Number of random independent simulations to perform. For each node and/or tip, there will be `Nsimulations` random states generated.
`drop_dims`	Logical, specifying whether singleton dimensions should be dropped from `tip_states` and `node_states`, if `Nsimulations==1` and/or Ntraits==1. If `drop_dims==FALSE`, then `tip_states` and `tip_nodes` will always be 3D matrices.

Details

The BM model for Ntraits co-evolving traits is defined by the stochastic differential equation

$$dX = \sigma \cdot dW$$

where W is a multidimensional Wiener process with Ndegrees independent components and σ is a matrix of size Ntraits x Ndegrees. Alternatively, the same model can be defined as a Fokker-Planck equation for the probability density ρ:

$$\frac{\partial \rho}{\partial t} = \sum_{i,j} D_{ij} \frac{\partial^2 \rho}{\partial x_i \partial x_j}.$$

The matrix D is referred to as the diffusivity matrix (or diffusion tensor), and $2D = \sigma \cdot \sigma^T$. Either `diffusivity` ($D$) or `sigma` ($\sigma$) may be used to specify the BM model, but not both.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). The asymptotic time complexity of this function is O(Nedges*Nsimulations*Ntraits).

Value

A list with the following elements:

`tip_states`	Either `NULL` (if `include_tips==FALSE`), or a 3D numeric matrix of size Nsimulations x Ntips x Ntraits. The [r,c,i]-th entry of this matrix will be the state of trait i at tip c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1` and Ntraits==1, then `tip_states` will be a vector.
`node_states`	Either `NULL` (if `include_nodes==FALSE`), or a 3D numeric matrix of size Nsimulations x Nnodes x Ntraits. The [r,c,i]-th entry of this matrix will be the state of trait i at node c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1` and Ntraits==1, then `node_states` will be a vector.

Author(s)

Stilianos Louca

See Also

`simulate_ou_model`, `simulate_rou_model`, `simulate_mk_model`, `fit_bm_model`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=10000)$tree

# Example 1: Scalar case
# - - - - - - - - - - - - - - - -
# simulate scalar continuous trait evolution on the tree
tip_states = simulate_bm_model(tree, diffusivity=1)$tip_states

# plot histogram of simulated tip states
hist(tip_states, breaks=20, xlab="state", main="Trait probability distribution", prob=TRUE)

# Example 2: Multivariate case
# - - - - - - - - - - - - - - - -
# simulate co-evolution of 2 traits with 3 degrees of freedom
Ntraits  = 2
Ndegrees = 3
sigma    = matrix(stats::rnorm(n=Ntraits*Ndegrees, mean=0, sd=1), ncol=Ndegrees)
tip_states = simulate_bm_model(tree, sigma=sigma, drop_dims=TRUE)$tip_states

# generate scatterplot of traits across tips
plot(tip_states[,1],tip_states[,2],xlab="trait 1",ylab="trait 2",cex=0.5)
```

simulate_deterministic_hbd

Simulate a deterministic homogenous birth-death model.

Description

Given a homogenous birth-death (HBD) model, i.e., with speciation rate λ, extinction rate μ and sampling fraction ρ, calculate various deterministic features of the model backwards in time, such as the total diversity over time. The speciation and extinction rates may be time-dependent. “Homogenous” refers to the assumption that, at any given moment in time, all lineages exhibit the same speciation/extinction rates (in the literature this is sometimes referred to simply as “birth-death model”; Morlon et al. 2011). “Deterministic” refers to the fact that all calculated properties are completely determined by the model’s parameters (i.e. non-random), as if an infinitely large tree was generated (aka. “continuum limit”).

Alternatively to λ, one may provide the pulled diversification rate (PDR; Louca et al. 2018) and the present-day speciation rate $\lambda(0)$. Similarly, alternatively to μ, one may provide the ratio of extinction over speciation rate, μ/λ. In either case, the time-profiles of λ, μ, μ/λ or the PDR are specified as piecewise polynomial functions (splines), defined on a discrete grid of ages.

Usage

```
simulate_deterministic_hbd(Ntips,
                           oldest_age,
                           rho            = 1,
                           age_grid       = NULL,
                           lambda         = NULL,
                           mu             = NULL,
                           mu_over_lambda = NULL,
                           PDR            = NULL,
                           lambda0        = NULL,
                           splines_degree = 1,
                           relative_dt    = 1e-3)
```

Arguments

`Ntips`	The assumed number of sampled extant species at present, defining the initial condition for the simulation. If the HBD model is supposed to describe a specific timetree, then Ntips should correspond to the number of tips in the tree.

`oldest_age`	Strictly positive numeric, specifying the oldest time before present ("age") to include in the simulation.
`rho`	Numeric between 0 (exclusive) and 1 (inclusive), specifying the sampling fraction ρ of the tree, i.e. the fraction of extant species included (aka. "rarefaction"). Note that if $rho < 1$, species are assumed to have been sampled randomly at equal probabilities. Can also be `NULL`, in which case $\rho = 1$ is assumed.
`age_grid`	Numeric vector, listing discrete ages (time before present) on which either λ and μ, or the PDR and μ, are specified. Listed ages must be strictly increasing, and must cover at least the full considered age interval (from 0 to `oldest_age`). Can also be `NULL` or a vector of size 1, in which case the speciation rate, extinction rate and PDR are assumed to be time-independent.
`lambda`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing speciation rates (λ, in units 1/time) at the ages listed in `age_grid`. Speciation rates should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then `PDR` and `lambda0` must be provided.
`mu`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing extinction rates (μ, in units 1/time) at the ages listed in `age_grid`. Extinction rates should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). Either `mu` or `mu_over_lambda` must be provided, but not both.
`mu_over_lambda`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing the ratio of extinction rates over speciation rates (μ/λ) at the ages listed in `age_grid`. These ratios should be non-negative, and are assumed to vary polynomially between grid points (see argument `splines_degree`). Either `mu` or `mu_over_lambda` must be provided, but not both.
`PDR`	Numeric vector, of the same size as `age_grid` (or size 1 if `age_grid==NULL`), listing pulled diversification rates (in units 1/time) at the ages listed in `age_grid`. PDRs can be negative or positive, and are assumed to vary polynomially between grid points (see argument `splines_degree`). If `NULL`, then `lambda` must be provided.
`lambda0`	Non-negative numeric, specifying the present-day extinction rate (in units 1/time). Either `lambda0` or `lambda` must be provided, but not both.
`splines_degree`	Integer, either 0,1,2 or 3, specifying the polynomial degree of the provided `lambda`, `mu` and `PDR` between grid points in `age_grid`. For example, if `splines_degree==1`, then the provided `lambda`, `mu` and `PDR` are interpreted as piecewise-linear curves; if `splines_degree==2` they are interpreted as quadratic splines; if `splines_degree==3` they are interpreted as cubic splines. The `splines_degree` influences the analytical properties of the curve, e.g. `splines_degree==1` guarantees a continuous curve, `splines_degree==2` guarantees a continuous curve and continuous derivative, and so on.
`relative_dt`	Strictly positive numeric (unitless), specifying the maximum relative time step allowed for integration over time. Smaller values increase integration accuracy but increase computation time. Typical values are 0.0001-0.001. The default is usually sufficient.

Details

This function supports the following alternative parameterizations of HBD models:

- Using the speciation rate λ and extinction rate μ.
- Using the speciation rate λ and the ratio μ/λ.
- Using the pulled diversification rate (PDR), the extinction rate and the present-day speciation rate $\lambda(0)$.
- Using the PDR, the ratio μ/λ and the present-day speciation rate $\lambda(0)$.

The PDR is defined as $PDR = \lambda - \mu + \lambda^{-1} d\lambda/d\tau$, where τ is age (time before present). To avoid ambiguities, only one of the above parameterizations is accepted at a time. The sampling fraction `rho` should always be provided; setting it to `NULL` is equivalent to setting it to 1.

Value

A named list with the following elements:

`success`	Logical, indicating whether the calculation was successful. If `FALSE`, then the returned list includes an additional '`error`' element (character) providing a description of the error; all other return variables may be undefined.
`ages`	Numerical vector of size NG, listing discrete ages (time before present) on which all returned time-curves are specified. Listed ages will be in ascending order, will cover exactly the range 0 - `oldest_age`, may be irregularly spaced, and may be finer than the original provided `age_grid`. Note that `ages[1]` corresponds to the present, while `ages[NG]` corresponds to the oldest time point (`oldest_age`).
`total_diversity`	Numerical vector of size NG, listing the predicted (deterministic) total diversity (number of extant species, denoted N) at the ages given in `ages[]`.
`shadow_diversity`	Numerical vector of size NG, listing the predicted (deterministic) "shadow diversity" at the ages given in `ages[]`. The shadow diversity is defined as $N_s = N \cdot \rho\lambda(0)/\lambda$.
`Pmissing`	Numeric vector of size NG, listing the probability that a lineage, extant at a given age, will be absent from the tree either due to extinction or due to incomplete sampling.
`Pextinct`	Numeric vector of size NG, listing the probability that a lineage, extant at a given age, will be fully extinct at present. Note that always `Pextinct<=Pmissing`.
`LTT`	Numeric vector of size NG, listing the number of lineages represented in the tree at any given age, also known as "lineages-through-time" (LTT) curve. Note that `LTT[1]` will be equal to Ntips, and that values in `LTT` will be non-increasing with age.
`lambda`	Numeric vector of size NG, listing the speciation rate (in units 1/time) at the ages given in `ages[]`.
`mu`	Numeric vector of size NG, listing the extinction rate (in units 1/time) at the ages given in `ages[]`.

`diversification_rate`	Numeric vector of size NG, listing the net diversification rate ($\lambda - \mu$) at the ages given in `ages[]`.
`PDR`	Numeric vector of size NG, listing the pulled diversification rate (PDR, in units 1/time) at the ages given in `ages[]`.
`PND`	Numeric vector of size NG, listing the pulled normalized diversity (PND, in units 1/time) at the ages given in `ages[]`. The PND is defined as $PND = (N/N(0)) \cdot \lambda(0)/\lambda$.
`SER`	Numeric vector of size NG, listing the "shadow extinction rate" (SER, in units 1/time) at the ages given in `ages[]`. The SER is defined as $SER = \rho\lambda(0) - PDR$.
`PER`	Numeric vector of size NG, listing the "pulled extinction rate" (PER, in units 1/time) at the ages given in `ages[]`. The PER is defined as $SER = \lambda(0) - PDR$ (Louca et al. 2018).
`PSR`	Numeric vector of size NG, listing the "pulled speciation rate" (PSR, in units 1/time) at the ages given in `ages[]`. The PSR is defined as $PSR = \lambda \cdot (1 - Pmissing)$.
`rholambda0`	Non-negative numeric, specifying the product of the sampling fraction and the present-day speciation rate, $\rho \cdot \lambda(0)$.

Author(s)

Stilianos Louca

References

H. Morlon, T. L. Parsons, J. B. Plotkin (2011). Reconciling molecular phylogenies with the fossil record. Proceedings of the National Academy of Sciences. 108:16327-16332.

S. Louca et al. (2018). Bacterial diversification through geological time. Nature Ecology & Evolution. 2:1458-1467.

See Also

`loglikelihood_hbd`

Examples

```
# define an HBD model with exponentially decreasing speciation/extinction rates
Ntips     = 1000
beta      = 0.01 # exponential decay rate of lambda over time
oldest_age= 10
age_grid  = seq(from=0,to=oldest_age,by=0.1) # choose a sufficiently fine age grid
lambda    = 1*exp(beta*age_grid) # define lambda on the age grid
mu        = 0.2*lambda # assume similarly shaped but smaller mu

# simulate deterministic HBD model
simulation = simulate_deterministic_hbd(Ntips,
                                        oldest_age,
                                                    rho         = 0.5,
                                                    age_grid    = age_grid,
                                                    lambda      = lambda,
                                                    mu          = mu)

# plot deterministic LTT
plot( x = simulation$ages, y = simulation$LTT, type='l',
      main='dLTT', xlab='age', ylab='lineages')
```

---

simulate_diversification_model
Simulate a deterministic uniform speciation/extinction model.

Description

Simulate a speciation/extinction cladogenic model for diversity over time, in the derministic limit. Speciation (birth) and extinction (death) rates can each be constant or power-law functions of the number of extant species. For example,

$$B = I + F \cdot N^E,$$

where B is the birth rate, I is the intercept, F is the power-law factor, N is the current number of extant species and E is the power-law exponent. Optionally, the model can account for incomplete taxon sampling (rarefaction of tips) and for the effects of collapsing a tree at a non-zero resolution (i.e. clustering closely related tips into a single tip).

Usage

```
simulate_diversification_model( times,
                                parameters              = list(),
                                added_rates_times       = NULL,
                                added_birth_rates_pc    = NULL,
                                added_death_rates_pc    = NULL,
                                added_periodic          = FALSE,
                                start_time              = NULL,
                                final_time              = NULL,
                                start_diversity         = 1,
                                final_diversity         = NULL,
                                reverse                 = FALSE,
                                include_coalescent      = FALSE,
                                include_event_rates     = FALSE,
                                include_Nevents         = FALSE,
                                max_runtime             = NULL)
```

Arguments

`times`	Numeric vector, listing the times for which to calculate diversities, as predicted by the model. Values must be in ascending order.

`parameters` A named list specifying the birth-death model parameters, with one or more of the following entries:

- `birth_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which new species (tips) are added. In units 1/time.
- `birth_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which new species (tips) are added. In units 1/time.
- `birth_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which new species (tips) are added. Unitless.
- `death_rate_intercept`: Non-negative number. The intercept of the Poissonian rate at which extant species (tips) go extinct. In units 1/time.
- `death_rate_factor`: Non-negative number. The power-law factor of the Poissonian rate at which extant species (tips) go extinct. In units 1/time.
- `death_rate_exponent`: Numeric. The power-law exponent of the Poissonian rate at which extant species (tips) go extinct. Unitless.
- `resolution`: Non-negative number. Time resolution at which the final tree is assumed to be collapsed. Units are time units. E.g. if this is 10, then all nodes of age 10 or less, are assumed to be collapsed into (represented by) a single tip. This can be used to model OTU trees, obtained after clustering strains by some similarity (=age) threshold. Set to 0 to disable collapsing. If left unspecified, this is set to 0.
- `rarefaction`: Numeric between 0 and 1, specifying the fraction of tips kept in the final tree after random subsampling. Rarefaction is assumed to occur after collapsing at the specified resolution (if applicable). This can be used to model incomplete taxon sampling. If left unspecified, this is set to 1.

`added_rates_times`
Numeric vector, listing time points (in ascending order) for a custom per-capita birth and/or death rates time series (see `added_birth_rates_pc` and `added_death_rates_pc` below). Can also be `NULL`, in which case the custom time series are ignored.

`added_birth_rates_pc`
Numeric vector of the same size as `added_rates_times`, listing per-capita birth rates to be added to the power law part. Added rates are interpolated linearly between time points in `added_rates_times`. Can also be `NULL`, in which case this option is ignored and birth rates are purely described by the power law.

`added_death_rates_pc`
Numeric vector of the same size as `added_rates_times`, listing per-capita death rates to be added to the power law part. Added rates are interpolated linearly between time points in `added_rates_times`. Can also be `NULL`, in which case this option is ignored and death rates are purely described by the power law.

`added_periodic`
Logical, indicating whether `added_birth_rates_pc` and `added_death_rates_pc` should be extended periodically if needed (i.e. if not defined for the entire simulation time). If `FALSE`, added birth & death rates are extended with zeros.

`start_time` Numeric. Start time of the tree (<=`times[1]`). Can also be `NULL`, in which case it is set to the first value in `times`.

`final_time`	Numeric. Final (ending) time of the tree (>=`max(times)`). Can also be `NULL`, in which case it is set to the last value in `times`.
`start_diversity`	Numeric. Total diversity at `start_time`. Only relevant if `reverse==FALSE`.
`final_diversity`	Numeric. Total diversity at `final_time`, i.e. the final diversity of the tree (total extant species at age 0). Only relevant if `reverse==TRUE`.
`reverse`	Logical. If `TRUE`, then the tree model is simulated in backward time direction. In that case, `final_diversity` is interpreted as the known diversity at the last time point, and all diversities at previous time points are calculated based on the model. If `FALSE`, then the model is simulated in forward-time, with initial diversity given by `start_diversity`.
`include_coalescent`	Logical, specifying whether the diversity corresponding to a coalescent tree (i.e. the tree spanning only extant tips) should also be calculated. If `coalescent==TRUE` and the death rate is non-zero, then the coalescent diversities will generally be lower than the total diversities.
`include_event_rates`	Logical. If `TRUE`, then the birth (speciation) and death (extinction) rates (for each time point) are included as returned values. This comes at a moderate computational overhead.
`include_Nevents`	Logical. If `TRUE`, then the cumulative birth (speciation) and death (extinction) events (for each time point) are included as returned values. This comes at a moderate computational overhead.
`max_runtime`	Numeric. Maximum runtime (in seconds) allowed for the simulation. If this time is surpassed, the simulation aborts.

Details

The simulation is deterministic, meaning that diversification is modeled using ordinary differential equations, not as a stochastic process. The simulation essentially computes the deterministic diversity over time, not an actual tree. For stochastic cladogenic simulations yielding a random tree, see `generate_random_tree` and `simulate_dsse`.

In the special case where per-capita birth and death rates are constant (i.e. $I = 0$ and $E = 1$ for birth and death rates), this function uses an explicit analytical solution to the underlying differential equations, and is thus much faster than in the general case.

If `rarefaction<1` and `resolution>0`, collapsing of closely related tips (at the resolution specified) is assumed to take place prior to rarefaction (i.e., subsampling applies to the already collapsed tips).

Value

A named list with the following elements:

`success`	Logical, indicating whether the simulation was successful. If the simulation aborted due to runtime constraints (option `max_runtime`), `success` will be `FALSE`.

`total_diversities`
: Numeric vector of the same size as `times`, listing the total diversity (extant at each the time) for each time point in `times`.

`coalescent_diversities`
: Numeric vector of the same size as `times`, listing the coalescent diversity (i.e. as seen in the coalescent tree spanning only extant species) for each time point in `times`. Only included if `include_coalescent==TRUE`.

`birth_rates`
: Numeric vector of the same size as `times`, listing the speciation (birth) rate at each time point. Only included if `include_event_rates==TRUE`.

`death_rates`
: Numeric vector of the same size as `times`, listing the extinction (death) rate at each time point. Only included if `include_event_rates==TRUE`.

`Nbirths`
: Numeric vector of the same size as `times`, listing the cumulative number of speciation (birth) events up to each time point. Only included if `include_Nevents==TRUE`.

`Ndeaths`
: Numeric vector of the same size as `times`, listing the cumulative number of extinction (death) events up to each time point. Only included if `include_Nevents==TRUE`.

Author(s)

Stilianos Louca

See Also

`generate_random_tree`, `count_lineages_through_time`

Examples

```
# Generate a tree
max_time = 100
parameters = list(birth_rate_intercept  = 10,
                  birth_rate_factor     = 0,
                  birth_rate_exponent   = 0,
                  death_rate_intercept  = 0,
                  death_rate_factor     = 0,
                  death_rate_exponent   = 0,
                  resolution            = 20,
                  rarefaction           = 0.5)
generator = generate_random_tree(parameters,max_time=max_time)
tree = generator$tree
final_total_diversity = length(tree$tip.label)+generator$Nrarefied+generator$Ncollapsed

# Calculate diversity-vs-time curve for the tree
times = seq(from=0,to=0.99*max_time,length.out=50)
tree_diversities = count_lineages_through_time(tree, times=times)$lineages

# simulate diversity curve based on deterministic model
simulation = simulate_diversification_model(times,
                                            parameters,
                                            reverse=TRUE,
                                            final_diversity=final_total_diversity,
                                            include_coalescent=TRUE)
```

```
model_diversities = simulation$coalescent_diversities

# compare diversities in the tree to the simulated ones
plot(tree_diversities,model_diversities,xlab="tree diversities",ylab="simulated diversities"
abline(a=0,b=1,col="#A0A0A0") # show diagonal for reference
```

---

simulate_dsse *Simulate a Discrete-State Speciation and Extinction (dSSE) model.*

Description

Simulate a random phylogenetic tree in forward time based on a Poissonian speciation/extinction (birth/death) process, whereby birth and death rates are determined by a co-evolving discrete trait. New species are added (born) by splitting of a randomly chosen extant tip. The discrete trait, whose values determine birth/death rates, can evolve in two modes: (A) Anagenetically, i.e. according to a discrete-space continuous-time Markov process along each edge, with fixed transition rates between states, and/or (B) cladogenetically, i.e. according to fixed transition probabilities between states at each speciation event. This model class includes the Multiple State Speciation and Extinction (MuSSE) model described by FitzJohn et al. (2009), as well as the Cladogenetic SSE (ClaSSE) model described by Goldberg and Igis (2012). Optionally, the model can be turned into a Hidden State Speciation and Extinction model (Beaulieu and O'meara, 2016), by replacing the simulated tip/node states with "proxy" states, thus hiding the original states actually influencing speciation/extinction rates.

Usage

```
simulate_dsse( Nstates,
               NPstates             = NULL,
               proxy_map            = NULL,
               parameters           = list(),
               start_state          = NULL,
               max_tips             = NULL,
               max_time             = NULL,
               max_time_eq          = NULL,
               sampling_fractions   = NULL,
               reveal_fractions     = NULL,
               coalescent           = TRUE,
               as_generations       = FALSE,
               no_full_extinction   = TRUE,
               Nsplits              = 2,
               tip_basename         = "",
               node_basename        = NULL,
               include_birth_times  = FALSE,
               include_death_times  = FALSE,
               include_rates        = FALSE,
               include_labels       = TRUE)
```

```
simulate_musse(Nstates,  NPstates = NULL, proxy_map = NULL, parameters = list(),
            start_state = NULL, max_tips = NULL, max_time = NULL, max_time_eq = NUI
            sampling_fractions = NULL, reveal_fractions = NULL, coalescent = TRUE,
            as_generations = FALSE, no_full_extinction = TRUE, Nsplits = 2,
            tip_basename = "", node_basename = NULL, include_birth_times = FALSE,
            include_death_times = FALSE, include_rates = FALSE, include_labels = TF
```

Arguments

`Nstates`	Integer, specifying the number of possible discrete states a tip can have, influencing speciation/extinction rates. For example, if `Nstates==2` then this corresponds to the common Binary State Speciation and Extinction (BiSSE) model (Maddison et al., 2007). In the case of a HiSSE model, `Nstates` refers to the total number of diversification rate categories, as described by Beaulieu and O'meara (2016).
`NPstates`	Integer, optionally specifying a number of "proxy-states" that are observed instead of the underlying speciation/extinction-modulating states. To simulate a HiSSE model, this should be smaller than `Nstates`. Each state corresponds to a different proxy-state, as defined using the variable `proxy_map` (see below). For BiSSE/MuSSE with no hidden states, `NPstates` can be set to either `NULL` or equal to `Nstates`, and proxy-states are equivalent to states.
`proxy_map`	Integer vector of size `Nstates` and with values in 1,..`NPstates`, specifying the correspondence between states (i.e. diversification-rate categories) and (observed) proxy-states, in a HiSSE model. Specifically, `proxy_map[s]` indicates which proxy-state the state s is represented by. Each proxy-state can represent multiple states (i.e. proxies are ambiguous), but each state must be represented by exactly one proxy-state. For non-HiSSE models, set this to `NULL`. See below for more details.
`parameters`	A named list specifying the dSSE model parameters, such as the anagenetic and/or cladogenetic transition rates between states and the state-dependent birth/death rates (see details below).
`start_state`	Integer within 1,..,`Nstates`, specifying the initial state, i.e. of the first lineage created. If left unspecified, this is chosen randomly and uniformly among all possible states.
`max_tips`	Maximum number of tips in the generated tree, prior to any subsampling. If `coalescent=TRUE`, this refers to the number of extant tips, prior to subsampling. Otherwise, it refers to the number of extinct + extant tips, prior to subsampling. If `NULL` or <=0, the number of tips is not limited, so you should use `max_time` and/or `max_time_eq` to stop the simulation.
`max_time`	Maximum duration of the simulation. If `NULL` or <=0, this constraint is ignored.
`max_time_eq`	Maximum duration of the simulation, counting from the first point at which speciation/extinction equilibrium is reached, i.e. when (birth rate - death rate) changed sign for the first time. If `NULL` or <0, this constraint is ignored.
`sampling_fractions`	A single number, or a numeric vector of size `NPstates`, listing tip sub-sampling fractions, depending on proxy-state. `sampling_fractions[p]` is the probability of including a tip in the final tree, if its proxy-state is p. If `NULL`, all tips

(or all extant tips, if `coalescent==TRUE`) are included in the tree. If a single number, all tips are included with the same probability, i.e. regardless of their proxy-state.

`reveal_fractions`
: Numeric vector of size `NPstates`, listing reveal fractions of tip proxy-states, depending on proxy state. `reveal_fractions[p]` is the probability of knowing a tip's proxy-state, if its proxy state is p. Can also be NULL, in which case all tip proxy states will be known.

`coalescent`
: Logical, specifying whether only the coalescent tree (i.e. the tree spanning the extant tips) should be returned. If `coalescent==FALSE` and the death rate is non-zero, then the tree may include non-extant tips (i.e. tips whose distance from the root is less than the total time of evolution). In that case, the tree will not be ultrametric.

`as_generations`
: Logical, specifying whether edge lengths should correspond to generations. If FALSE, then edge lengths correspond to time.

`no_full_extinction`
: Logical, specifying whether to prevent complete extinction of the tree. Full extinction is prevented by temporarily disabling extinctions whenever the number of extant tips is 1. if `no_full_extinction==FALSE` and death rates are non-zero, the tree may go extinct during the simulation; if `coalescent==TRUE`, then the returned tree would be empty, hence the function will return unsuccessfully (i.e. `success` will be `FALSE`). By default `no_full_extinction` is `TRUE`, however in some special cases it may be desirable to allow full extinctions to ensure that the generated trees are statistically distributed exactly according to the underlying cladogenetic model.

`Nsplits`
: Integer greater than 1. Number of child-tips to generate at each diversification event. If set to 2, the generated tree will be bifurcating. If >2, the tree will be multifurcating.

`tip_basename`
: Character. Prefix to be used for tip labels (e.g. "tip."). If empty (""), then tip labels will be integers "1", "2" and so on.

`node_basename`
: Character. Prefix to be used for node labels (e.g. "node."). If `NULL`, no node labels will be included in the tree.

`include_birth_times`
: Logical. If `TRUE`, then the times of speciation events (in order of occurrence) will also be returned.

`include_death_times`
: Logical. If `TRUE`, then the times of extinction events (in order of occurrence) will also be returned.

`include_rates`
: Logical. If `TRUE`, then the per-capita birth & death rates of all tips and nodes will also be returned.

`include_labels`
: Logical, specifying whether to include tip-labels and node-labels (if available) as names in the returned state vectors (e.g. `tip_states` and `node_states`). In any case, returned states are always listed in the same order as tips and nodes

in the tree. Setting this to FALSE may increase computational efficiency for situations where labels are not required.

Details

The function simulate_dsse can be used to simulate a diversification + discrete-trait evolutionary process, in which birth/death (speciation/extinction) rates at each tip are determined by a tip's current "state". Lineages can transition between states anagenetically along each edge (according to fixed Markov transition rates) and/or cladogenetically at each speciation event (according to fixed transition probabilities).

The function simulate_musse is a simplified variant meant to simulate MuSSE/HiSSE models in the absence of cladogenetic state transitions, and is included mainly for backward-compatibility reasons. The input arguments for simulate_musse are identical to simulate_dsse, with the exception that the parameters argument must include slightly different elements (explained below).

For simulate_dsse, the argument parameters should be a named list including one or more of the following elements:

- birth_rates: Numeric vector of size Nstates, listing the per-capita birth rate (speciation rate) at each state. Can also be a single number (all states have the same birth rate).
- death_rates: Numeric vector of size Nstates, listing the per-capita death rate (extinction rate) at each state. Can also be a single number (all states have the same death rate).
- transition_matrix_A: 2D numeric matrix of size Nstates x Nstates, listing anagenetic transition rates between states along an edge. Hence, transition_matrix_A[r,c] is the probability rate for transitioning from state r to state c. Non-diagonal entries must be non-negative, diagonal entries must be non-positive, and the sum of each row must be zero.
- transition_matrix_C: 2D numeric matrix of size Nstates x Nstates, listing cladogenetic transition rates between states during a speciation event, seperately for each child. Hence, transition_matrix_C[r,c] is the probability that a child will have state c, conditional upon the occurrence of a speciation event, given that the parent had state r, and independently of all other children. Entries must be non-negative, and the sum of each row must be one.

For simulate_musse, the argument parameters should be a named list including one or more of the following elements:

- birth_rates: Same as for simulate_dsse.
- death_rates: Same as for simulate_dsse.
- transition_matrix: 2D numeric matrix of size Nstates x Nstates, listing anagenetic transition rates between states. This is equivalent to transition_matrix_A in simulate_dsse.

If max_time==NULL and max_time_eq==NULL, then the returned tree will always contain max_tips tips. In particular, if at any moment during the simulation the tree only includes a single extant tip, the death rate is temporarily set to zero to prevent the complete extinction of the tree. If max_tips==NULL, then the simulation is ran as long as specified by max_time and/or max_time_eq. If neither max_time, max_time_eq nor max_tips is NULL, then the simulation halts as soon as the time exceeds max_time, or the time since equilibration exceeds max_time_eq, or the number of tips (extant tips if coalescent is TRUE) exceeds max_tips, whichever occurs first. If max_tips!=NULL and Nsplits>2, then the last diversification even may generate fewer than `Nsplits` children, in order to keep the total number of tips within the specified limit. Note that this code generates trees in forward time, and halts as soon as one of the halting conditions is met; the halting condition chosen affects the precise distribution from which the generated trees are drawn (Stadler 2011).

HiSSE models (Beaulieu and O'meara, 2016) are closely related to BiSSE/MuSSE models, the main difference being the fact that the actual diversification-modulating states are not directly observed. Hence, this function is also able to simulate HiSSE models, with appropriate choice of the input variables `Nstates`, `NPstates` and `proxy_map`. For example, `Nstates=4`, `NPstates=2` and `proxy_map=c(1,2,1,2)` specifies that states 1 and 3 are represented by proxy-state 1, and states 2 and 4 are represented by proxy-state 2. This is the original case described by Beaulieu and O'meara (2016); in their terminology, there would be 2 "hidden"" states ("0" and "1") and 2 "observed" (proxy) states ("A" and "B"), and the 4 diversification rate categories (`Nstates=4`) would be called "0A", "1A", "0B" and "1B", respectively. The somewhat different terminology used here allows for easier generalization to an arbitrary number of diversification-modulating states and an arbitrary number of proxy states. For example, if there are 6 diversification modulating states, represented by 3 proxy-states as 1->A, 2->A, 3->B, 4->C, 5->C, 6->C, then one would set `Nstates=6`, `NPstates=3` and `proxy_map=c(1,1,2,3,3,3)`.

The parameter `transition_matrix_C` can be used to define ClaSSE models (Goldberg and Igic, 2012) or BiSSE-ness models (Magnuson-Ford and Otto, 2012), although care must be taken to properly define the transition probabilities. Here, cladogenetic transitions occur at probabilities that are defined conditionally upon a speciation event, whereas in other software they may be defined as probability rates.

Value

A named list with the following elements:

`success`	Logical, indicating whether the simulation was successful. If `FALSE`, an additional element `error` (of type character) is included containing an explanation of the error; in that case the value of any of the other elements is undetermined.
`tree`	A rooted bifurcating (if `Nsplits==2`) or multifurcating (if `Nsplits>2`) tree of class "phylo", generated according to the specified birth/death model. If `coalescent==TRUE` or if all death rates are zero, and only if `as_generations==FALSE`, then the tree will be ultrametric. If `as_generations==TRUE` and `coalescent==FALSE`, all edges will have unit length.
`root_time`	Numeric, giving the time at which the tree's root was first split during the simulation. Note that if `coalescent==TRUE`, this may be later than the first speciation event during the simulation.
`final_time`	Numeric, giving the final time at the end of the simulation. If `coalescent==TRUE`, then this may be greater than the total time span of the tree (since the root of the coalescent tree need not correspond to the first speciation event).
`equilibrium_time`	Numeric, giving the first time where the sign of (death rate - birth rate) changed from the beginning of the simulation, i.e. when speciation/extinction equilibrium was reached. May be infinite if the simulation stopped before reaching this point.

`Nbirths`	Numeric vector of size Nstates, listing the total number of birth events (speciations) that occurred at each state. The sum of all entries in `Nbirths` may be lower than the total number of tips in the tree if death rates were non-zero and `coalescent==TRUE`, or if `Nsplits>2`.
`Ndeaths`	Numeric vector of size Nstates, listing the total number of death events (extinctions) that occurred at each state.
`Ntransitions_A`	2D numeric matrix of size Nstates x Nstates, listing the total number of anagenetic transition events that occurred between each pair of states. For example, `Ntransitions_A[1,2]` is the number of anagenetic transitions (i.e., within a species) that occured from state 1 to state 2.
`Ntransitions_C`	2D numeric matrix of size Nstates x Nstates, listing the total number of cladogenetic transition events that occurred between each pair of states. For example, `Ntransitions_C[1,2]` is the number of cladogenetic transitions (i.e., from a parent to a child) that occured from state 1 to state 2 during some speciation event. Note that each speciation event will have caused `Nsplits` transitions, and that the emergence of a child with the same state as the parent is counted as a transition between the same state (diagonal entries in `Ntransitions_C`).
`tip_states`	Integer vector of size Ntips and with values in 1,..,Nstates, listing the state of each tip in the tree.
`node_states`	Integer vector of size Nnodes and with values in 1,..,Nstates, listing the state of each node in the tree.
`tip_proxy_states`	Integer vector of size Ntips and with values in 1,..,NPstates, listing the proxy state of each tip in the tree. Only included in the case of HiSSE models.
`node_proxy_states`	Integer vector of size Nnodes and with values in 1,..,NPstates, listing the proxy state of each node in the tree. Only included in the case of HiSSE models.
`start_state`	Integer, specifying the state of the first lineage (either provided during the function call, or generated randomly).
`birth_times`	Numeric vector, listing the times of speciation events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`death_times`	Numeric vector, listing the times of extinction events during tree growth, in order of occurrence. Note that if `coalescent==TRUE`, then `speciation_times` may be greater than the phylogenetic distance to the coalescent root.
`clade_birth_rates`	Numeric vector of size Ntips+Nnodes, listing the per-capita birth rate of each tip and node in the tree. Only included if `include_rates==TRUE`.
`clade_death_rates`	Numeric vector of size Ntips+Nnodes, listing the per-capita death rate of each tip and node in the tree. Only included if `include_rates==TRUE`.

Author(s)

Stilianos Louca

References

W. P. Maddison, P. E. Midford, S. P. Otto (2007). Estimating a binary character's effect on speciation and extinction. Systematic Biology. 56:701-710.

R. G. FitzJohn, W. P. Maddison, S. P. Otto (2009). Estimating trait-dependent speciation and extinction rates from incompletely resolved phylogenies. Systematic Biology. 58:595-611

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

E. E. Goldberg, B. Igic (2012). Tempo and mode in plant breeding system evolution. Evolution. 66:3701-3709.

K. Magnuson-Ford, S. P. Otto (2012). Linking the investigations of character evolution and species diversification. The American Naturalist. 180:225-245.

J. M. Beaulieu and B. C. O'Meara (2016). Detecting hidden diversification shifts in models of trait-dependent speciation and extinction. Systematic Biology. 65:583-601.

T. Stadler (2011). Simulating trees with a fixed number of extant species. Systematic Biology. 60:676-684.

See Also

`fit_musse`

Examples

```
# Simulate a tree under a BiSSE model (i.e., anagenetic transitions between two states)
A = get_random_mk_transition_matrix(Nstates=2, rate_model="ER", max_rate=0.1)
parameters = list(birth_rates         = c(1,1.5),
                  death_rates         = 0.5,
                  transition_matrix_A = A)
simulation = simulate_dsse( Nstates       = 2,
                            parameters    = parameters,
                            max_tips      = 1000,
                            include_rates = TRUE)
tree       = simulation$tree
Ntips      = length(tree$tip.label)

# plot distribution of per-capita birth rates of tips
rates = simulation$clade_birth_rates[1:Ntips]
barplot(table(rates)/length(rates),
        xlab="rate",
        main="Distribution of pc birth rates across tips (BiSSE model)")
```

`simulate_mk_model` *Simulate an Mk model for discrete trait evolution.*

Description

Given a rooted phylogenetic tree, a fixed-rates continuous-time Markov model for the evolution of a discrete trait ("Mk model", described by a transition matrix) and a probability vector for the root, simulate random outcomes of the model on all nodes and/or tips of the tree. The function traverses nodes from root to tips and randomly assigns a state to each node or tip based on its parent's previously assigned state and the specified transition rates between states. The generated states have joint distributions consistent with the Markov model. Optionally, multiple independent simulations can be performed using the same model.

Usage

```
simulate_mk_model(tree, Q, root_probabilities="stationary",
                  include_tips=TRUE, include_nodes=TRUE,
                  Nsimulations=1, drop_dims=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`Q`	A numeric matrix of size Nstates x Nstates, storing the transition rates between states. In particular, every row must sum up to zero.
`root_probabilities`	Probabilities of the different states at the root. Either a character vector with value "stationary" or "flat", or a numeric vector of length Nstates, where Nstates is the number of possible states of the trait. In the later case, `root_probabilities` must be a valid probability vector, i.e. with non-negative values summing up to 1. "stationary" sets the probabilities at the root to the stationary distribution of `Q` (see `get_stationary_distribution`), while "flat" means that each state is equally probable at the root.
`include_tips`	Include random states for the tips. If `FALSE`, no states will be returned for tips.
`include_nodes`	Include random states for the nodes. If `FALSE`, no states will be returned for nodes.
`Nsimulations`	Number of random independent simulations to perform. For each node and/or tip, there will be `Nsimulations` random states generated.
`drop_dims`	Logical, specifying whether the returned `tip_states` and `node_states` (see below) should be vectors, if `Nsimulations==1`. If `drop_dims==FALSE`, then `tip_states` and `tip_nodes` will always be 2D matrices.

Details

For this function, the trait's states must be represented by integers within 1,..,Nstates, where Nstates is the total number of possible states. If the states are originally in some other format (e.g. characters or factors), you should map them to a set of integers 1,..,Nstates. These integers should correspond to row & column indices in the transition matrix `Q`. You can easily map any set of discrete states to integers using the function `map_to_state_space`.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). The time required per simulation decreases with the total number of requested simulations.

Value

A list with the following elements:

`tip_states`	Either `NULL` (if `include_tips==FALSE`), or a 2D integer matrix of size Nsimulations x Ntips with values in 1,..,Nstates, where Ntips is the number of tips in the tree and Nstates is the number of possible states of the trait. The [r,c]-th entry of this matrix will be the state of tip c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `tip_states` will be a vector.
`node_states`	Either `NULL` (if `include_nodes==FALSE`), or a 2D integer matrix of size Nsimulations x Nnodes with values in 1,..,Nstates, where Nnodes is the number of nodes in the tree. The [r,c]-th entry of this matrix will be the state of node c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `node_states` will be a vector.

Author(s)

Stilianos Louca

See Also

`exponentiate_matrix`, `get_stationary_distribution`, `simulate_bm_model`, `simulate_ou_model`, `simulate_rou_model`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=1000)$tree

# simulate discrete trait evolution on the tree (5 states)
Nstates = 5
Q = get_random_mk_transition_matrix(Nstates, rate_model="ARD", max_rate=0.1)
tip_states = simulate_mk_model(tree, Q)$tip_states

# plot histogram of simulated tip states
barplot(table(tip_states)/length(tip_states), xlab="state")
```

simulate_ou_model *Simulate an Ornstein-Uhlenbeck model for continuous trait evolution.*

Description

Given a rooted phylogenetic tree and an Ornstein-Uhlenbeck (OU) model for the evolution of a continuous (numeric) trait, simulate random outcomes of the model on all nodes and/or tips of the tree. The function traverses nodes from root to tips and randomly assigns a state to each node or tip based on its parent's previously assigned state and the specified model parameters. The generated states have joint distributions consistent with the OU model. Optionally, multiple independent simulations can be performed using the same model.

Usage

```
simulate_ou_model(tree, stationary_mean, spread, decay_rate,
                  include_tips=TRUE, include_nodes=TRUE,
                  Nsimulations=1, drop_dims=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`stationary_mean`	Numeric. The mean (center) of the stationary distribution of the OU model.
`spread`	Numeric. The standard deviation of the stationary distribution of the OU model.
`decay_rate`	Numeric. Exponential decay rate (stabilization rate) of the OU model (in units 1/edge_length_units).
`include_tips`	Include random states for the tips. If `FALSE`, no states will be returned for tips.
`include_nodes`	Include random states for the nodes. If `FALSE`, no states will be returned for nodes.
`Nsimulations`	Number of random independent simulations to perform. For each node and/or tip, there will be `Nsimulations` random states generated.
`drop_dims`	Logical, specifying whether the returned `tip_states` and `node_states` (see below) should be vectors, if `Nsimulations==1`. If `drop_dims==FALSE`, then `tip_states` and `tip_nodes` will always be 2D matrices.

Details

For each simulation, the state of the root is picked randomly from the stationary distribution of the OU model, i.e. from a normal distribution with mean = `stationary_mean` and standard deviation = `spread`.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). The asymptotic time complexity of this function is O(Nedges*Nsimulations), where Nedges is the number of edges in the tree.

Value

A list with the following elements:

`tip_states`	Either `NULL` (if `include_tips==FALSE`), or a 2D numeric matrix of size Nsimulations x Ntips, where Ntips is the number of tips in the tree. The [r,c]-th entry of this matrix will be the state of tip c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `tip_states` will be a vector.
`node_states`	Either `NULL` (if `include_nodes==FALSE`), or a 2D numeric matrix of size Nsimulations x Nnodes, where Nnodes is the number of nodes in the tree. The [r,c]-th entry of this matrix will be the state of node c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `node_states` will be a vector.

Author(s)

Stilianos Louca

See Also

`simulate_bm_model`, `simulate_mk_model`, `simulate_rou_model`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=10000)$tree

# simulate evolution of a continuous trait
tip_states = simulate_ou_model(tree, stationary_mean=10, spread=1, decay_rate=0.1)$tip_state

# plot histogram of simulated tip states
hist(tip_states, breaks=20, xlab="state", main="Trait probability distribution", prob=TRUE)
```

`simulate_rou_model` *Simulate a reflected Ornstein-Uhlenbeck model for continuous trait evolution.*

Description

Given a rooted phylogenetic tree and a reflected Ornstein-Uhlenbeck (ROU) model for the evolution of a continuous (numeric) trait, simulate random outcomes of the model on all nodes and/or tips of the tree. The ROU process is similar to the Ornstein-Uhlenbeck process (see `simulate_ou_model`), with the difference that the ROU process cannot fall below a certain value (its "reflection point"), which (in this implementation) is also its deterministic equilibrium point (Hu et al. 2015). The function traverses nodes from root to tips and randomly assigns a state to each node or tip based on its parent's previously assigned state and the specified model parameters. The generated states have joint distributions consistent with the ROU model. Optionally, multiple independent simulations can be performed using the same model.

Usage

```
simulate_rou_model(tree, reflection_point, spread, decay_rate,
                   include_tips=TRUE, include_nodes=TRUE,
                   Nsimulations=1, drop_dims=TRUE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`reflection_point`	Numeric. The reflection point of the ROU model. In castor, this also happens to be the deterministic equilibrium of the ROU process (i.e. if the decay rate were infinite). For example, if a trait can only be positive (but arbitrarily small), then `reflection_point` may be set to 0.
`spread`	Numeric. The stationary standard deviation of the corresponding unreflected OU process.
`decay_rate`	Numeric. Exponential decay rate (stabilization rate) of the ROU process (in units 1/edge_length_units).
`include_tips`	Include random states for the tips. If `FALSE`, no states will be returned for tips.
`include_nodes`	Include random states for the nodes. If `FALSE`, no states will be returned for nodes.
`Nsimulations`	Number of random independent simulations to perform. For each node and/or tip, there will be `Nsimulations` random states generated.
`drop_dims`	Logical, specifying whether the returned `tip_states` and `node_states` (see below) should be vectors, if `Nsimulations==1`. If `drop_dims==FALSE`, then `tip_states` and `tip_nodes` will always be 2D matrices.

Details

For each simulation, the state of the root is picked randomly from the stationary distribution of the ROU model, i.e. from a one-sided normal distribution with mode = `reflection_point` and standard deviation = `stationary_std`.

If `tree$edge.length` is missing, each edge in the tree is assumed to have length 1. The tree may include multi-furcations (i.e. nodes with more than 2 children) as well as mono-furcations (i.e. nodes with only one child). The asymptotic time complexity of this function is O(Nedges*Nsimulations), where Nedges is the number of edges in the tree.

Value

A list with the following elements:

`tip_states`	Either `NULL` (if `include_tips==FALSE`), or a 2D numeric matrix of size Nsimulations x Ntips, where Ntips is the number of tips in the tree. The [r,c]-th entry of this matrix will be the state of tip c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `tip_states` will be a vector.

`node_states`	Either `NULL` (if `include_nodes==FALSE`), or a 2D numeric matrix of size Nsimulations x Nnodes, where Nnodes is the number of nodes in the tree. The [r,c]-th entry of this matrix will be the state of node c generated by the r-th simulation. If `drop_dims==TRUE` and `Nsimulations==1`, then `node_states` will be a vector.

Author(s)

Stilianos Louca

References

Y. Hu, C. Lee, M. H. Lee, J. Song (2015). Parameter estimation for reflected Ornstein-Uhlenbeck processes with discrete observations. Statistical Inference for Stochastic Processes. 18:279-291.

See Also

`simulate_ou_model`, `simulate_bm_model`, `simulate_mk_model`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=10000)$tree

# simulate evolution of a continuous trait whose value is always >=1
tip_states = simulate_rou_model(tree, reflection_point=1, spread=2, decay_rate=0.1)$tip_stat

# plot histogram of simulated tip states
hist(tip_states, breaks=20, xlab="state", main="Trait probability distribution", prob=TRUE)
```

`tree_distance` *Calculate the distance between two trees.*

Description

Given two rooted phylogenetic trees with identical tips, calculate their difference using a distance metric.

Usage

```
tree_distance(treeA, treeB, tipsA2B=NULL, metric="RF", normalized=FALSE)
```

Arguments

`treeA`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`treeB`	A rooted tree of class "phylo", with the same number of tips as `treeA`. The root is assumed to be the unique node with no incoming edge.

`tipsA2B`	Optional integer vector of size Ntips, mapping `treeA` tip indices to `treeB` tip indices (i.e. `tipsA2B[a]` is the tip index in `treeB` corresponding to tip index `a` in `treeA`). The mapping must be one-to-one. If left unspecified, it is determined by matching tip labels between the two trees (this assumes that the same tip labels are used in both trees).
`metric`	Character, specifying the distance metric to be used. Currently only the Robinson-Foulds ("RF") metric is implemented, which is the number of clusters (sets of tips descending from a node) in either of the trees but not shared by both trees (Robinson and Foulds, 1981; Day, 1985). The Robinson-Foulds metric does not take into account branch lengths.
`normalized`	Logical, specifying whether the calculated distance should be normalized to be between 0 and 1. For the Robinson-Foulds metric, the distance will be normalized by dividing it by the total number of nodes in the two trees.

Details

If the trees differ in theis tips, they must be pruned down to their common set of tips. If tips have different labels in the two trees, but are nevertheless equivalent, the mapping between the two trees must be provided using `tipsA2B`. The trees may include multi-furcations as well as mono-furcations (i.e. nodes with only one child).

Note that under some Robinson-Foulds variants the trees can be unrooted; in this present implementation trees must be rooted and the placement of the root influences the distance, following the definition by Day (1985).

The asymptotic average time complexity of this function is O(Nedges*log(Nedges)*log(log(Nedges))) for a balanced bifurcating tree.

Value

A single non-negative number, representing the distance between the two trees.

Author(s)

Stilianos Louca

References

Robinson, D. R., Foulds, L. R. (1981). Comparison of phylogenetic trees. Mathematical Biosciences. 53: 131-147.

Day, W. H. E. (1985). Optimal algorithms for comparing trees with labeled leaves. Journal of Classification. 2:7-28.

See Also

`congruent_divergence_times`

Examples

```
# generate a random tree
Ntips = 1000
treeA = generate_random_tree(list(birth_rate_intercept=1),
                             max_tips=Ntips)$tree

# create a second tree with slightly different topology
treeB = treeA
shuffled_tips = sample.int(Ntips, size=Ntips/10, replace=FALSE)
treeB$tip.label[shuffled_tips] = treeB$tip.label[sample(shuffled_tips)]

# calculate Robinson-Foulds distance between trees
distance = tree_distance(treeA, treeB, metric="RF")
```

---

trim_tree_at_height

Trim a rooted tree down to a specific height.

Description

Given a rooted phylogenetic tree and a maximum allowed distance from the root ("height"), remove tips and nodes and shorten the remaining terminal edges so that the tree's height does not exceed the specified threshold. This corresponds to drawing the tree in rectangular layout and trimming everything beyond a specific phylogenetic distance from the root. Tips or nodes at the end of trimmed edges are kept, and the affected edges are shortened.

Usage

```
trim_tree_at_height(tree, height = Inf, by_edge_count = FALSE)
```

Arguments

`tree`	A rooted tree of class "phylo". The root is assumed to be the unique node with no incoming edge.
`height`	Numeric, specifying the phylogenetic distance from the root at which to trim.
`by_edge_count`	Logical. Instead of considering edge lengths, consider edge counts as phylogenetic distance. This is the same as if all edges had length equal to 1.

Details

The input tree may include multi-furcations (i.e. nodes with more than 2 children) as well as monofurcations (i.e. nodes with only one child).

Tip labels and uncollapsed node labels of the collapsed tree are inheritted from the original tree. Labels of tips that used to be nodes (i.e. of which all descendants have been removed) will be the node labels from the original tree. If the input tree has no node names, it is advised to first add node names to avoid `NA` in the resulting tip names.

Value

A list with the following elements:

`tree`	A new rooted tree of class "phylo", representing the trimmed tree.
`Nedges_trimmed`	Integer. Number of edges trimmed (shortened).
`Nedges_removed`	Integer. Number of edges removed.
`new2old_clade`	Integer vector of length equal to the number of tips+nodes in the trimmed tree, with values in 1,..,Ntips+Nnodes, mapping tip/node indices of the trimmed tree to tip/node indices in the original tree. In particular, `c(tree$tip.label,tree$node.label)[new2old_clade]` will be equal to: `c(trimmed_tree$tip.label,trimmed_tree$node.label)`.
`new2old_edge`	Integer vector of length equal to the number of edges in the trimmed tree, with values in 1,..,Nedges, mapping edge indices of the trimmed tree to edge indices in the original tree. In particular, `tree$edge.length[new2old_edge]` will be equal to `trimmed_tree$edge.length` (if edge lengths are available).

Author(s)

Stilianos Louca

Examples

```
# generate a random tree, include node names
tree = generate_random_tree(list(birth_rate_intercept=1),
                            max_time=1000,
                            node_basename="node.")$tree

# print number of tips
cat(sprintf("Simulated tree has %d tips\n",length(tree$tip.label)))

# trim tree at height 500
trimmed = trim_tree_at_height(tree, height=500)$tree

# print number of tips in trimmed tree
cat(sprintf("Trimmed tree has %d tips\n",length(trimmed$tip.label)))
```

`write_tree` *Write a tree in Newick (parenthetic) format.*

Description

Write a phylogenetic tree to a file or a string, in Newick (parenthetic) format. If the tree is unrooted, it is first rooted internally at the first node.

Usage

```
write_tree(tree, file="", append=FALSE, digits=10)
```

Arguments

`tree`	A tree of class "phylo".
`file`	An optional path to a file, to which the tree should be written. The file may be overwritten without warning. If left empty (default), then a string is returned representing the tree.
`append`	Logical, specifying whether the tree should be appended at the end of the file, rather than replacing the entire file (if it exists).
`digits`	Number of significant digits for writing edge lengths.

Details

This function is comparable to (but typically much faster than) the `ape` function `write.tree`.

Value

If `file==""`, then a string is returned containing the Newick representation of the tree. Otherwise, the tree is directly written to the file and no value is returned.

Author(s)

Stilianos Louca

See Also

`read_tree`

Examples

```
# generate a random tree
tree = generate_random_tree(list(birth_rate_intercept=1),max_tips=100)$tree

# obtain a string representation of the tree in Newick format
Newick_string = write_tree(tree)
```