Package: corHMM (via r-universe)

October 16, 2026

Version 2.10.5

Date 2026-07-14

Title Hidden Markov Models of Character Evolution

Maintainer Jeremy Beaulieu <jmbeauli@uark.edu>

Depends ape, nloptr, GenSA

Suggests testthat, knitr, rmarkdown, dentist, Rtsne, dbscan, proxy, Rcpp, formatR

Imports expm, numDeriv, corpcor, MASS, nnet, phangorn, parallel, viridis, Rmpfr, igraph, phytools, lhs, RTMB, Matrix

Description Fits hidden Markov models of discrete character evolution which allow different transition rate classes on different portions of a phylogeny. Beaulieu et al (2013) <doi:10.1093/sysbio/syt034>.

License GPL (>= 2)

Remotes bomeara/dentist

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/pak/sysreqs cmake libglpk-dev libgmp3-dev libxml2-dev libmpfr-dev

Repository https://thej022214.r-universe.dev

Date/Publication 2026-10-16 15:43:42 UTC

RemoteUrl https://github.com/thej022214/corhmm

RemoteRef HEAD

RemoteSha 03a93238fb5a72dc3c70fb4157327436fb2b20ce

Contents

ancRECON 2
ancRECON_slice 5
compute_joint_ci 6
ComputeCI 8

ConvertPhangornReconstructions . . . 9
corDISC . . . 10
corHMM . . . 12
corHMMDredge . . . 17
corHMMDredgeBase . . . 23
dent_likelihood . . . 26
dent_propose . . . 26
dent_walk . . . 27
examples . . . 29
fitCorrelationTest . . . 29
get_batch_profile_lik . . . 31
getCVTable . . . 33
getFullMat . . . 33
getModelTable . . . 35
getStateMat4Dat . . . 36
kFoldCrossValidation . . . 38
makeSimmap . . . 39
node_importance_by_cluster . . . 42
phylo_aware_dist_fast . . . 43
phylo_aware_dist_local . . . 44
plot.dentist . . . 46
plot_batch_profile_lik . . . 46
plot_clustered_phylo . . . 48
plot_raw_recon . . . 49
plot_stacked_densities . . . 51
plot_transition_summary . . . 52
plotDredgeTrace . . . 53
plotMKmodel . . . 54
plotRECON . . . 56
print.dentist . . . 57
rayDISC . . . 57
simMarkov . . . 61
summarize_transition_stats . . . 62
summary.dentist . . . 63
tipHomology . . . 64

Index **66**

`ancRECON` *Ancestral state reconstruction*

Description

Infers ancestral states based on a set of model parameters

Usage

```
ancRECON(phy,data, p, method=c("joint", "marginal", "scaled"),
rate.cat, ntraits=NULL, rate.mat=NULL,
model="ARD", root.p=NULL, get.likelihood=FALSE, get.tip.states = FALSE,
tip.fog=NULL, get.info=FALSE, collapse = TRUE)
```

Arguments

phy	a phylogenetic tree, in ape "phylo" format.
data	a data matrix containing species information (see Details).
p	a vector of transition rates to be used to estimate ancestral states.
method	method used to calculate ancestral states at internal nodes. Can be one of: "joint", "marginal", or "scaled" (see Details).
rate.cat	specifies the number of rate categories in the HRM.
ntraits	currently, this is automaticall detected and can always be set to NULL.
rate.mat	a user-supplied rate matrix index of parameters to be optimized.
model	specifies the underlying model if a rate.mat is not provided ("ER", SYM", or "ARD").
root.p	a vector used to fix the probabilities at the root, but "yang" and "maddfitz" can also be supplied to use the method of Yang (2006) and FitzJohn et al (2009) respectively (see details).
get.likelihood	a logical indicating whether to obtain the likelihood of the rates and states. The default is FALSE.
get.tip.states	a logical indicating whether just tip reconstructions should be output. The default is FALSE.
tip.fog	provides the probability that an observed state is not actually in the state it is assigned to the reconstruction algorithm. These values are assumed either optimized in "corHMM" or supplied by the user. tip.fog=NULL when using this as a standalone function.
get.info	Whether to return information measures at nodes (Boyko and Beaulieu 2021).
collapse	a boolean indicating whether to collapse multiple character combinations into only the observed states. For example, if true a two character dataset contained (0,0), (1,0), and (1,1), this would be collapsed into 1,2,3. However, if set to false it would 1,2,4. In combination with a custom rate matrix this allows for the estimation of transitions between the unobserved character combination. The default is TRUE

.

Details

This is a stand alone function for computing the marginal, joint, or scaled likelihoods of internal nodes for a given set of transition rates. Like all other functions contained in corHMM, the tree does not have to be bifurcating in order for analyses to be carried out. IMPORTANT: If the corDISC, corHMM, and rayDISC functions are used they automatically provide a tree with the likeliest states

as internal node labels. This function is intended for circumstances where the user would like to reconstruct states based on rates estimated elsewhere (e.g. BayesTraits, Mesquite, ape).

The algorithm based on Pupko et al. (2000, 2002) is used to calculate the `joint` estimates of ancestral states. The `marginal` method was originally implemented based on a description of an algorithm by Yang (2006). The basic idea is that the tree is rerooted on each internal node, with the marginal likelihood being the probabilities of observing the tips states given that the focal node is the root. However, this takes a ton of time as the number of nodes increase. But, importantly, this does not work easily when the model contains asymmetric rates. Here, we use the same dynamic programming algorithm as Mesquite (Maddison and Maddison, 2011), which is time linear with the number of species and calculates the marginal probability at a node using an additional up and down pass of the tree. If `scaled`, the function uses the same algorithm from ace(). Note that the `scaled` method of ace() is simply the conditional likelihoods of observing everything at or above the focal node and these should NOT be used for ancestral state estimation.

The user can fix the root state probabilities by supplying a vector to `root.p`. For example, in the two trait case, if the hypothesis is that the root is 00, then the root vector would be `root.p=c(1,0,0,0)` for state combinations 00, 01, 10, and 11, respectively. If analyzing a binary or multistate character, the order of root.p is the same order as the traits – e.g., for states 1, 2, 3, a `root.p=c(0,1,0)` would fix the root to be in state 2. If the user supplies the flag `root.p=`"yang", then the estimated transition rates are used to set the weights at the root (see pg. 124 Yang 2006), whereas specifying `root.p=`"maddfitz" employs the same procedure described by Maddison et al. (2007) and FitzJohn et al. (2009). Note that the default `root.p=NULL` assumes equal weighting among all possible states.

Setting get.likelihood=TRUE will provide the user the joint likelihood of the rates and states.

Value

`$lik.tip.states`
: A matrix of the reconstructed tip values. If the number of rate.cats is greater than 2 then the probability that each observed state is in a particular hidden state is given.

`$lik.anc.states`
: For `joint`, a vector of likeliest states at internal nodes and tips. For either `marginal` or `$scaled`, a matrix of the probabilities of each state for each internal node are returned.

`$info.anc.states`
: A vector containing the amount of information (in bits) that the tip states and model gives to each node. See Boyko and Beaulieu (2021).

Author(s)

Jeremy M. Beaulieu and Jeffrey C. Oliver

References

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

Maddison, W.P. and D.R. Maddison. 2011. Mesquite: a modular system for evolutionary analysis. Version 2.75 http://mesquiteproject.org

Pupko, T., I. Pe'er, R. Shamir, and D. Graur. 2000. A fast algorithm for joint reconstruction of ancestral amino-acid sequences. Molecular Biology and Evolution 17:890-896.

Pupko, T., I. Pe'er, D. Graur, M. Hasegawa, and N Friedman N. 2002. A branch-and-bound algorithm for the inference of ancestral amino-acid sequences when the replacement rate varies among sites: application to the evolution of five gene families. Bioinformatics 18:1116-1123.

Yang, Z. 2006. Computational Molecular Evolution. London:Oxford.

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Examples

```
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
MK_3state <- corHMM(phy = phy, data = data, rate.cat = 1)

# # one way to get the parameters from your corHMM object in the correct order
p <- sapply(1:max(MK_3state$index.mat, na.rm = TRUE), function(x)
 na.omit(c(MK_3state$solution))[na.omit(c(MK_3state$index.mat) == x)][1])

# using custom params
states_1 <- ancRECON(phy = phy, data = MK_3state$data, p = p, method = "marginal",
 rate.cat <- MK_3state$rate.cat, ntraits = NULL, rate.mat = MK_3state$index.mat,
 root.p = MK_3state$root.p)
```

---

ancRECON_slice *Ancestral state reconstruction for a particular time*

Description

Infers marginal ancestral states based on a set of model parameters at a particular time

Usage

```
ancRECON_slice(corhmm.obj, time_slice, edge=NULL, collapse=TRUE, ncores = 1)
```

Arguments

corhmm.obj	a corhmm object which is the output of the main corhmm function.
time_slice	a vector of times to reconstruct (present = 0, root = max(branching.times(phy)))
edge	specifies specific edges to reconstruct only. default is NULL meaning all edges along a slice are reconstructed.
collapse	set collapse to be the same as it was during the corhmm run.
ncores	number of cores to use during parallelization.

Details

This is a stand alone function for computing the marginal likelihoods at particular points along a branch for a given set of transition rates. ancRECON has the technical details of ancestral state reconstruction if you're interested. The time_slice argument will specify a time point where marginal reconstructions will be produced. You can imagine the time slice intersecting the branches of the phylogeny and doing a reconstruction there rather than at nodes as is typically done.

Value

a data.frame. Each row of time_slice is the time period that was specified. node is the closest tipward node to the slice on a particular branch. position is the amount of time (rootward) from the nearest node. Remaining columns are the marginal probabilities of each state at that particular node. There is also a plotting function that I don't currently export because it's unfinished. The example shows how to access it.

Author(s)

James D. Boyko

References

Yang, Z. 2006. Computational Molecular Evolution. London:Oxford.

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Examples

```
library(corHMM)
library(viridis)
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
corhmm.obj <- corHMM(phy = phy, data = data, rate.cat = 1)
test <- ancRECON_slice(corhmm.obj, time_slice = c(1, 10, 20, 30, 40, 49),
         collapse = TRUE, ncores = 1)
 corHMM:::plot_slice_recon(phy, test, col = viridis(3))
```

---

compute_joint_ci *Compute joint ancestral-state reconstructions by sampling*

Description

Compute joint ancestral state reconstructions (ASR) by sampling joint histories and returning a description of their likelihoods. The function first precomputes the necessary transition structure, then samples joint histories using one of several methods (simmap, marginal, or Pupko-based sampling), until a maximum number of samples is reached. The result is a list containing the best joint history, the corresponding log-likelihood, and the ensemble of sampled histories with their log-likelihoods, suitable for downstream uncertainty analyses.

Usage

```
compute_joint_ci(res, batch_size = 100, max_samples = 1000,
remove_outliers = TRUE, p_method = "simmap")
```

Arguments

`res` A corHMM results object. The function uses internal components of res, including:

- `res$phy`: a phylogenetic tree (class `"phylo"`).
- `res$data`: a data.frame of observed traits.
- `res$collapse`: boolean indicating whether states were collapsed.
- `res$rate.cat`: number of rate categories.
- `res$index.mat`: index matrix describing parameter groupings for the model.
- `res$root.p`: root prior (vector or method flag).

`batch_size` Integer. Number of joint histories to sample per iteration. Default is 100.

`max_samples` Integer. Maximum total number of joint histories to sample. Default is 1000.

`remove_outliers` Logical. If `TRUE`, remove outliers in the log-likelihood distribution using a boxplot-based rule. Default is `TRUE`.

`p_method` Character. Sampling strategy. One of `"simmap"` (default), `"marginal"`, or `"pupko"`.

Details

This function implements a two-pass sampling approach to quantify the uncertainty of joint ancestral-state reconstructions. Key steps include:

- Computing p from the input result object and constructing the Q-matrix, then precomputing the transition probability matrices for each edge length.
- Determining the best joint history and its log-likelihood.
- Obtaining a conditional likelihood object for stochastic mapping-based sampling.
- Iteratively sampling batches of joint histories using the specified `p_method`:
  - `"simmap"`: sample by simulating substitution histories with conditional likelihoods.
  - `"marginal"`: sample marginal reconstructions and combine them into joint histories.
  - `"pupko"`: use Pupko-based sampling of joint histories.
- Aggregating sampled histories, removing duplicates, and filtering out (negative) infinite ln-likelihoods.
- Optionally removing outlier ln-likelihoods to improve stability.
- Assembling `state_df` and `tip_state_df` from the remaining histories.

Value

A list with components:

- `best_joint:` The internal representation of the best joint history as returned by `ancRECON_internal(..., method = "joint", get.likelihood = FALSE)`.
- `best_lnlik:` The log-likelihood corresponding to `best_joint` (obtained with `get.likelihood = TRUE`).
- `sampled_joints:` A list of sampled joint histories. Each element is expected to contain at least `anc.states` (internal node states) and `tip.states` (tip states).
- `state_df:` Data frame of internal node states across the sampled histories.
- `tip_state_df:` Data frame of tip states across the sampled histories.
- `lnliks:` Numeric vector of log-likelihoods for the sampled histories.

Author(s)

James D. Boyko

Examples

```
 ## Not run:
## res is a corHMM-style result object produced earlier in your workflow
ci <- compute_joint_ci(res)
str(ci)

## End(Not run)
```

ComputeCI *Compute confidence around rate estimates*

Description

corHMM gives a single point estimate of rates, but this point estimate could be very uncertain. A traditional way to evaluate confidence is to vary each parameter until the log likelihood gets 2 units worse, while holding other parameters at their maximum likelihood estimates. That's fine and fast, but it can be misled by ridges. So, instead, we want all values that lead to a likelihood within two log likelihood units of the best. The range will be at least as wide as the univariate estimates but probably much larger.

Usage

```
ComputeCI(corhmm.object, desired.delta = 2, n.points=5000, verbose=TRUE,
print_freq=50, ...)
```

Arguments

`corhmm.object`	The result of a corHMM search.
`desired.delta`	How many log likelihood units to deviate from the optimal likelihood.
`n.points`	How many points to use.
`print_freq`	Output progress every print_freq steps.
`verbose`	Other arguments to pass into the likelihood function.
`...`	further arguments to be passed dentist.

Details

The algorithm tunes: if it is moving too far away from the desired likelihoods, it will decrease the proposal width; if it staying in areas better than the desired likelihood, it will increase the proposal width. It will also expand the proposal width for parameters where the extreme values still appear good enough to try to find out the full range for these values.

In general, the idea of this is not to give you a pleasingly narrow range of possible values – it is to try to find the actual uncertainty, including finding any ridges that would not be seen in univariate space.

Value

A dentist object containing results, the data.frame of negative log likelihoods and the parameters associated with them; acceptances, the vector of whether a proposed move was accepted each step; best_neglnL, the best value passed into the analysis; delta, the desired offset; all_ranges, a summary of the results.

Author(s)

Brian O'Meara (see also the R package Dentist) & exported by James D. Boyko

---

`ConvertPhangornReconstructions` *Convert phangorn reconstruction to a vector*

Description

Converts a character reconstruction from phangorn into a vector of tip and node states. Nodes where there are equal weights among states, ties are broken at random.

Usage

```
ConvertPhangornReconstructions(x, site = 1, best = TRUE)
```

Arguments

x	The phyDat object that contains a character reconstruction from phangorn
site	The character number to convert into a vector
best	A logical indicating whether the state that maximizes some function (likelihood, parsimony, etc.) is to be returned.

Details

Creates a vector that contains the best tips and node state from a phangorn reconstruction.

---

corDISC *Correlated evolution binary traits*

Description

Fits a model of correlated evolution between two or three binary traits

Usage

```
corDISC(phy,data, ntraits=2, rate.mat=NULL, model=c("ER","SYM","ARD"),
node.states=c("joint", "marginal", "scaled", "none"), lewis.asc.bias=FALSE, p=NULL,
root.p=NULL, ip=NULL, lb=0, ub=100, diagn=FALSE)
```

Arguments

phy	a phylogenetic tree, in ape "phylo" format.
data	a data matrix containing species information (see Details).
ntraits	specifies the number of traits to be included in the analysis.
rate.mat	a user-supplied rate matrix index of parameters to be optimized.
model	specifies the underlying model.
node.states	method used to calculate ancestral states at internal nodes (see Details).
lewis.asc.bias	a logical indicating whether the ascertainment bias correction of Lewis et al. 2001 should be used. The default is FALSE.
p	a vector of transition rates. Allows the user to calculate the likelihood given a specified set of parameter values to specified as fixed and calculate the likelihood.
root.p	a vector used to fix the probabilities at the root, but "yang" and "maddfitz" can also be supplied to use the method of Yang (2006) and FitzJohn et al (2009) respectively (see details).
ip	initial values used for the likelihood search. Can be a single value or a vector of unique values for each parameter. The default is ip=1.
lb	lower bound for the likelihood search. The default is lb=0.
ub	upper bound for the likelihood search. The default is ub=100.
diagn	logical indicating whether diagnostic tests should be performed. The default is FALSE.

Details

__THIS FUNCTION IS NO LONGER NECESSARY AS IT IS NOW ENTIRELY SUBSUMED WITHIN__ `corHMM` (see _Generalized corHMM_ vignette). But we still provide it for those that are more comfortable using it than exploring the new `corHMM` function. As before, `corDISC` takes a tree and a trait file and estimates transition rates and ancestral states for two or three binary characters (see Pagel 1994). Note, however, that rayDISC can be used to evaluate the same models as in corDISC, with the major difference being that, with rayDISC, the rate matrix would have to be manipulated using `rate.mat.maker` in order to remove parameters associated with dual transitions. With corDISC, the input phylogeny need not be bifurcating as the algorithm is implemented to handle multifucations. Polytomies are allowed by generalizing Felsenstein's (1981) pruning algorithm to be the product of the probability of observing the tip states of n descendant nodes, rather than two, as in the completely bifurcating case. For the trait file, the first column of the trait file must contain the species labels to match to the tree, with the second column onwards corresponding to the binary traits of interest.

The user can fix the root state probabilities by supplying a vector to `root.p`. For example, in the two trait case, if the hypothesis is that the root is 00, then the root vector would be `root.p=c(1,0,0,0)` for state combinations 00, 01, 10, and 11, respectively. If the user supplies the flag `root.p="yang"`, then the estimated transition rates are used to set the weights at the root (see pg. 124 Yang 2006), whereas specifying `root.p="maddfitz"` employs the same procedure described by Maddison et al. (2007) and FitzJohn et al. (2009). Note that the default `root.p=NULL` assumes equal weighting among all possible states.

We also note that scoring information that is missing for a species can be incorporated in the analysis by including an NA for that particular trait. corDISC will then set the trait vector so that the tip vector will reflect the probabilities that are compatible with our observations. For example, if the scoring for trait 1 is missing, but trait 2 is scored as 0, then the tip vector would be (1,0,1,0), for state combinations 00, 01, 10, and 11 respectively, given our observation that trait 2 is scored 0 (for a good discussion see Felsenstein 2004, pg. 255).

Value

`corDISC` returns an object of class `corDISC`. This is a list with elements:

- `$loglik` — the maximum negative log-likelihood.
- `$AIC` — Akaike information criterion.
- `$AICc` — Akaike information criterion corrected for sample size.
- `$ntraits` — The number of traits specified.
- `$solution` — a matrix containing the maximum likelihood estimates of the transition rates.
- `$solution.se` — a matrix containing the approximate standard errors of the transition rates. The standard error is calculated as the square root of the diagonal of the inverse of the Hessian matrix.
- `$index.mat` — The indices of the parameters being estimated are returned. The numbers correspond to the row in the `eigvect` and can useful for identifying the parameters that are causing the objective function to be at a saddlepoint.
- `$lewis.asc.bias` — The setting describing whether or not the Lewis ascertainment bias correction was used.

`$opts`	Internal settings of the likelihood search
`$data`	User-supplied dataset.
`$phy`	User-supplied tree.
`$states`	The likeliest states at each internal node.
`$tip.states`	NULL
`$iterations`	The number of iterations used by the optimization routine.
`$eigval`	The eigenvalues from the decomposition of the Hessian of the likelihood function. If any `eigval<0` then one or more parameters were not optimized during the likelihood search
`$eigvect`	The eigenvectors from the decomposition of the Hessian of the likelihood function is returned

Author(s)

Jeremy M. Beaulieu

References

Beaulieu J.M., and M.J. Donoghue 2013. Fruit evolution and diversification in campanulid angiosperms. Evolution, 67:3132-3144.

Felsenstein, J. 1981. A likelihood approach to character weighting and what it tells us about parsimony and compatibility. Biological Journal of the Linnean Society 16: 183-196.

Felsenstein J. 2004. Inferring phylogenies. Sunderland MA: Sinauer Associates.

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

Lewis, P.O. 2001. A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology 50:913-925.

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

Pagel, M. 1994. Detecting correlated evolution on phylogenies: a general method for the comparative analysis of discrete characters. Proceedings of the Royal Society, B. 255:37-45.

`corHMM` *Hidden Rates Model*

Description

Estimates hidden rates underlying the evolution of any number of multi-state characters

Usage

```
corHMM(phy, data, rate.cat, rate.mat=NULL, model = "ARD",
node.states = "marginal", fixed.nodes=FALSE, p=NULL, root.p="yang",
tip.fog=NULL, ip=NULL, fog.ip=0.01, nstarts=0, n.cores=1,
get.tip.states = FALSE, lewis.asc.bias = FALSE, collapse = TRUE,
lower.bound = 1e-9, upper.bound = 100, opts=NULL, return.devfun = FALSE, use_RTMB = FALSE,
verbose=TRUE)
```

Arguments

`phy`	a phylogenetic tree, in `ape` "phylo" format.
`data`	a data.frame containing species information. The first column must be species names matching the phylogeny. Additional columns contain discrete character data. All columns are converted to class factor unless already is.factor. Order of states is determined by the levels of column. Unobserved levels may be modeled if collapse=FALSE.
`rate.cat`	specifies the number of hidden rate categories (if 1, there are no hidden states, see Details).
`rate.mat`	a user-supplied matrix containing indexes of parameters to be optimized (see Details).
`model`	One of "ARD", "SYM", or "ER". ARD: all rates differ. SYM: rates between any two states do not differ. ER: all rates are equal.
`node.states`	method used to calculate ancestral states at internal nodes (see Details).
`fixed.nodes`	specifies that states for nodes in the phylogeny are assumed fixed. These are supplied as node labels in the "phylo" object.
`p`	a vector of transition rates. Allows the user to calculate the likelihood given a specified set of parameter values to specified as fixed and calculate the likelihood (see Details).
`root.p`	a vector used to fix the probabilities at the root, but "yang" and "maddfitz" can also be supplied to use the method of Yang (2006) and FitzJohn et al (2009) respectively (see details).
`tip.fog`	a fixed value or vector of free parameters to estimate the probability that an observed state is not actually in the state it is assigned. By default `tip.fog=NULL`. To estimate tip fog probabilities a vector of integers designating the number of free parameters is needed as input. See vignette for details on how to carry this out.
`ip`	initial values used for the likelihood search. Can be a single value or a vector of unique values for each parameter. The default is `ip=1`.
`fog.ip`	initial values used for the likelihood search of tip fog. It is always a single value. The default is `fog.ip=0.01`.
`nstarts`	the number of random restarts to be performed. The default is `nstarts=0`.
`n.cores`	the number of processor cores to spread out the random restarts.
`get.tip.states`	a boolean indicating whether tip reconstructions should be output. The default is `FALSE`.

`lewis.asc.bias` a boolean indicating whether to correct for observing a dataset that is not univariate. The default is `FALSE`

.

`collapse` a boolean indicating whether to collapse multiple character combinations into only the observed states. For example, if true a two character dataset contained (0,0), (1,0), and (1,1), this would be collapsed into 1,2,3. However, if set to false it would 1,2,4. In combination with a custom rate matrix this allows for the estimation of transitions between the unobserved character combination. The default is `TRUE`

.

`lower.bound` lower bound for the likelihood search. The default is `lower.bound=1e-9`.

`upper.bound` upper bound for the likelihood search. The default is `upper.bound=100`.

`opts` options to pass to nloptr. default is `NULL`.

`return.devfun` return deviance function?

`use_RTMB` use RTMB for optimization?

`verbose` a boolean indicating whether to print progress to screen. default is `TRUE`

Details

This function takes a tree and a trait file and estimates transition rates and ancestral states for any number of discrete characters using a Markov model with or without "hidden" states. Users are advised to read the _Generalized corHMM_ vignette for details on how to make full use of `corHMM`'s new functionality. In general, these models describe evolution as discrete transitions between observed states. If `rate.cat` > 1, then the model is a hidden Markov model (HMM; also known as a hidden rates model (HRM)). The HRM is a generalization of the covarion model that allows different rate classes to be treated as "hidden" states. Essentially a hidden Markov model allows for multiple processes to describe the evolution of your observed character. This could be another (hidden) state or a large group of them. Regardless of the reason, an HMM is saying that not all observed characters are expected to act the same way.

The first column of the input data must be species names (as in the previous version), but there can be any number of data columns. If your dataset does have 2 or more columns of trait information, each column is taken to describe a separate character. The separation of character and state is an important one because corHMM will automatically remove dual transitions from your model. For example, say you had 3 characters each with 2 states (0 or 1), but only three of these combinations were ever observed 0_0_1, 0_1_0, or 1_0_0. With dual transitions disallowed, it is impossible to move between these combinations because it would mean simultaneously losing and gaining a state (0_0_1 -> 0_0_0 -> 0_1_0 in one step.) One way around this is to provide a custom rate matrix to corHMM where transitions are allowed between these states. However, this is also a case where it would seem appropriate to code the data as a single character with 3 states.

Ambiguities (polymorphic taxa or taxa missing data) are assigned likelihoods following Felsenstein (2004, p. 255). Taxa with missing data are coded "?" with all states observed at a tip. Polymorphic taxa are coded with states separated by an "&". For example, if a trait has four states and taxonA is observed to be in state 1 and 3, the character would be coded as "1&3". corHMM then uses this information to assign a likelihood of 1.0 to both states. Missing data are treated as ambiguous for

all states, thus all states for taxa missing data are assigned a likelihood of 1.0. For example, for a four-state character (i.e. DNA), a taxon missing data will have likelihoods of all four states equal to 1.0 [e.g. L(A)=1.0, L(C)=1.0, L(G)=1.0, L(T)=1.0].

Rate matrices are usually generated via the specified `model`, but custom matrices can be used by supplying `rate.mat`. Similar to the output of `model`, the required format is a matrix of dimensions nTraitStates x nTraitStates, in which each transition is given an index value. Transitions with the same value will be optimised to have the same rate, so e.g. a vector with all 1's would correspond to the ER model. Transitions which are not allowed (e.g. representing two changes at once, like 00 -> 11) or not possible (e.g. the diagonal) can be set as 0 or `NA`.

If a user wants to calculate the likelihood for a fixed set of transition rates, without optimising them, a vector `p` of those values has to be provided. The positon of those values in the vector should correspond the the index numbers in `rate.mat`, if provided, or to those resulting from the specified `model`. If for any reason, any index numbers are not present in `rate.mat`, those positions in the vector will have to be populated with a placeholder (any number or `NA` will do), to prevent rates from shifting when being assigned to the transitions.

The likelihood function is maximized using the bounded subplex optimization routine implemented in the R package `nloptr`, which provides a common interface to NLopt, an open-source library for nonlinear optimization. In the former case, however, it is recommended that `nstarts` is set to a large value (e.g. 100) to ensure that the maximum likelihood solution is found. Users can set `n.cores` to parse the random restarts onto multiple processors.

The user can fix the root state probabilities by supplying a vector to `root.p`. For example, if the hypothesis is that the root is 0_S in a model with two hidden rates (their categories being denoted "S" and "F" respectively), then the root vector would be `root.p=c(1,0,0,0)` for state combinations 0_S, 1_S, 0_F, and 1_F, respectively. If the user supplies the flag `root.p=`"NULL", then there is equal weighting among all possible states in the model. If the user supplies the flag `root.p=`"yang", then the estimated transition rates are used to set the weights at the root (see pg. 124 Yang 2006), whereas specifying `root.p=`"maddfitz" employs the same procedure described by Maddison et al. (2007) and FitzJohn et al. (2009). Note that the default `root.p="yang"`.

Ancestral states can be estimated using marginal, joint, scaled, or none approaches. Marginal gives the likelihood of state at each node, integrating over the states at other nodes. Joint gives the optimal state at each node for the entire tree at once (it can only return the most likely state, i.e. it is not a probability like the marginal reconstruction). Scaled is included for compatibility with ape's ace() function. None suppresses calculation of ancestral states, which can dramatically speed up calculations if you're comparing models but make plotting difficult.

Value

`corHMM` returns an object of class `corHMM`. This is a list with elements:

`$loglik`	the maximum negative log-likelihood.
`$AIC`	Akaike information criterion.
`$AICc`	Akaike information criterion corrected for sample size.
`$rate.cat`	The number of rate categories specified.
`$solution`	a matrix containing the maximum likelihood estimates of the transition rates. Note that the rate classes are ordered from slowest (R1) to fastest (Rn) with respect to state 0.

`$index.mat`	The indices of the parameters being estimated are returned. This also is a way to allow the estimation of transition rates for parameters not oberved in the dataset. Say you have 2 traits X and Y, where the combinations 00, 01, and 11 are observed (10 is not). A 4 by 4 index matrix could be used to force 10 into the model.
`$data`	User-supplied dataset.
`$data.legend`	User-supplied dataset with an extra column of trait values corresponding to how corHMM calls the user data.
`$phy`	User-supplied tree.
`$states`	The likeliest states at each internal node. The state and rates reconstructed at internal nodes are in the order of the column headings of the rates matrix.
`$tip.states`	The likeliest state at each tip. The state and rates reconstructed at the tips are in the order of the column headings of the rates matrix.
`$states.info`	a vector containing the amount of information (in bits) that the tip states and model gives to each node.
`$iterations`	The number of iterations used by the optimization routine.
`$root.p`	The root prior used in model estimation.
`pen.type`	The penalty type used in model estimation.
`lambda`	The hyperparameter that adjusts penalty severity used in model estimation.

Author(s)

Jeremy M. Beaulieu and James D. Boyko

References

Beaulieu J.M., B.C. O'Meara, and M.J. Donoghue. 2013. Identifying hidden rate changes in the evolution of a binary morphological character: the evolution of plant habit in campanulid angiosperms. Systematic Biology 62:725-737.

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Felsenstein, J. 1981. A likelihood approach to character weighting and what it tells us about parsimony and compatibility. Biological Journal of the Linnean Society 16: 183-196.

Felsenstein J. 2004. Inferring phylogenies. Sunderland MA: Sinauer Associates.

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

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

Pagel, M. 1994. Detecting correlated evolution on phylogenies: a gneeral method for the comparative analysis of discrete characters. Proc. R. Soc. Lond. B 255:37-45.

Yang, Z. 2006. Computational Molecular Evolution. Oxford Press:London.

Examples

```
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
MK_3state <- corHMM(phy = phy, data = data, rate.cat = 1)
MK_3state


### custom rate.mat and user supplied transition rates not to be optimised

# this is the rate matrix used in the above example, which represents the ARD
#model with four different rates for all four possible transitions
MK_3state$index.mat

# now we specify our own one where the transitions in and out of state 3 are
#forced to be the same:
rate.mat <- matrix(c(NA, 1, NA, 2, NA, 3, NA, 3, NA), 3, 3)

# we supply the fixed transition rates for which we want the likelihood
#calculated. Since we optimise 3 states in the above matrix, we have to
#give a vector with three rates, which will be applied to the transition
#with the corresponding rate above (e.g. the transition 2 -> 1 happens at
#rate 0.05)
p <- c(0.05, 0.02, 0.03)

# run
MK_3state.fixed <- corHMM(phy = phy, data = data, rate.cat = 1, rate.mat = rate.mat, p = p)
MK_3state.fixed
```

`corHMMDredge` *Automatic Discovery of Optimal Discrete Character Models via Simulated Annealing*

Description

This function automates the search for the best-fitting model of discrete character evolution. It uses a combination of simulated annealing and optional regularization to explore a vast space of possible model structures, identifying optimal models without requiring the user to manually specify and compare candidate models.

Usage

```
corHMMDredge(phy, data, max.rate.cat=1, init.rate.cat=1,
  root.p="maddfitz", tip.fog=NULL, fog.ip=0.01, pen.type="l1", lambda=0,
  drop.threshold=1e-7, criterion="AIC", merge.threshold=0, index_mat=NULL,
  node.states="none", fixed.nodes=FALSE, ip=NULL, nstarts=1, n.cores=1,
  get.tip.states=FALSE, lewis.asc.bias=FALSE, collapse=FALSE,
  lower.bound=1e-10, upper.bound=100, opts=NULL, verbose=TRUE,
```

```
p=NULL, rate.cat=NULL, use_RTMB=TRUE, max.iterations=1000,
initial.temp=2, cooling.rate=0.95, temp.schedule="exponential",
seed=NULL, require.entry=FALSE, stall.limit=500, polish.restarts=2,
polish.max.fits=5,
checkpoint.file=NULL, checkpoint.interval=50)
```

Arguments

`phy`	A phylogenetic tree of class `"phylo"`.
`data`	A data frame with two or more columns: the first containing species names and the second onwards containing the discrete character states.
`max.rate.cat`	The maximum number of hidden rate categories to explore. The search runs sequentially from one rate category up to this value.
`init.rate.cat`	Ignored. The search always begins at one rate category so that the simplest model is always fit and reported.
`root.p`	Specifies the prior on the root state probabilities. Can be `"maddfitz"`, `"yang"`, a numeric vector of probabilities, or `NULL` (see `corHMM`).
`tip.fog`	Optional specification of tip observation error (misclassification probabilities). If a numeric value less than 1 is supplied it is treated as a fixed per-state fog probability; if `sum(tip.fog) >= 1` the fog probabilities are estimated during optimisation.
`fog.ip`	Starting value for tip fog probability estimation. Defaults to 0.01.
`pen.type`	The type of regularization penalty to apply. Options are `"l1"` (lasso), `"l2"` (ridge), and `"er"` (equal rates penalty). Defaults to `"l1"`.
`lambda`	A hyper-parameter (0 to 1) controlling the strength of the regularization penalty. A value of 0 means no penalization.
`drop.threshold`	The rate value below which a parameter becomes a candidate for being dropped (fixed to zero) during a drop move.
`criterion`	The information criterion used to evaluate and compare models during the search. One of `"AIC"`, `"AICc"`, or `"BIC"`. Defaults to `"AIC"`.
`merge.threshold`	The absolute difference between two rate estimates below which they become candidates for merging (constrained equal) during a merge move.
`index_mat`	An optional user-specified starting rate matrix and transition mask. If `NULL` the search starts from a fully parameterized model using the transitions inferred from the data. An `NA` cell can never be freed; a non-`NA` off-diagonal cell is eligible even when it represents a simultaneous transition. Diagonal cells must be `NA`.
`node.states`	The method for ancestral state reconstruction. Options are `"marginal"`, `"joint"`, `"scaled"`, or `"none"`. Defaults to `"none"` during the search for speed; set to `"marginal"` if ancestral states are needed on the returned models.
`fixed.nodes`	Logical. Whether to fix the states of labelled internal nodes in the provided phylogeny.
`ip`	Optional numeric vector of initial parameter values for the optimiser.

`nstarts` The number of random restarts to perform during optimisation for each candidate model. Higher values improve the chance of finding the global likelihood peak at the cost of computation time.

`n.cores` The number of cores to use for parallel random restarts within a single model fit.

`get.tip.states` Logical. Whether to estimate marginal state probabilities at the tips.

`lewis.asc.bias` Logical. Whether to correct for ascertainment bias following Lewis (2001).

`collapse` Logical. Whether to collapse unobserved character state combinations.

`lower.bound`, `upper.bound` Lower and upper bounds for rate parameter estimates.

`opts` A list of control options passed to the `nloptr` optimiser.

`verbose` Logical. Whether to print search progress to the console.

`p`, `rate.cat` Used together to evaluate a single fixed model. If both are provided the dredge search is skipped and a single model fit is returned immediately.

`use_RTMB` Logical. Whether to use the RTMB automatic differentiation backend for gradient-based optimisation. Defaults to `TRUE`. When `TRUE` the gradient is computed via autodifferentiation, which is substantially faster for complex models.

`max.iterations` The maximum number of simulated annealing iterations within each rate category.

`initial.temp` The starting temperature for the simulated annealing algorithm. Higher values allow broader exploration by accepting worse models with higher probability early in the search.

`cooling.rate` The multiplicative rate at which temperature determines the target final temperature before scaling the cooling schedule across the iteration budget. Must be between 0 and 1. Defaults to 0.95.

`temp.schedule` The cooling schedule. Options are `"exponential"`, `"linear"`, and `"logarithmic"`. The selected schedule spans the available iteration budget.

`require.entry` If `TRUE` states with no incoming rates are treated as invalid. Defaults to `FALSE`, which permits irreversible models with absorbing states.

`seed` An integer random seed for reproducibility.

`stall.limit` Number of iterations without a new best score before deterministic neighbor polishing runs. Defaults to 500.

`polish.restarts` Number of bounded reheats allowed after polishing finds no improving neighbor. Defaults to 2.

`polish.max.fits` Maximum number of uncached candidate models fitted during one polishing pass. Defaults to 5. Cached neighbors do not count against the limit.

`checkpoint.file` Optional path to an RDS file used for checkpointing. If the file already exists the search resumes from the saved state; otherwise a new checkpoint is created at regular intervals. When `max.rate.cat` > 1 a separate checkpoint file is written per rate category (e.g. `ckpt_rc1.rds`, `ckpt_rc2.rds`).

`checkpoint.interval` Number of iterations between checkpoint saves. Defaults to 50.

Details

This function implements an automated model discovery framework for phylogenetic comparative methods. The core of the method is a simulated annealing heuristic that explores the space of discrete character evolution models combined with optional regularization.

Proposal moves. At each iteration the algorithm proposes one of five modifications to the current model's rate matrix index structure:

1. **Drop:** A rate parameter is removed from the model (fixed to zero), every cell carrying it at once. Parameters with smaller estimated values are preferentially targeted. If the drop strands a whole hidden class that class is removed and the rate category falls by one.
2. **Merge:** Two or more rate parameters are constrained to be equal. Pairs with similar estimated values are preferentially targeted. Rates sitting at the optimizer floor are excluded: the data calls them zero, so they belong to a drop rather than a merge.
3. **Free:** A currently constrained or dropped parameter is freed to be estimated independently, expanding model complexity[1].
4. **Lump:** An exactly lumpable pair of states (Kemeny and Snell) is found by checking whether the fitted rates from each state into every other state agree, and the corresponding parameters are equated. Because the likelihood is unchanged such a move needs no refit and lowers the selected criterion through its parameter penalty.
5. **Collapse:** All rows and columns belonging to a dead hidden class – one the rest of the model cannot reach – are deleted and the rate category is decremented. Only dead classes are eligible, since no move ever adds a class back.

   [1] Only positions permitted by the maximum index matrix are eligible, preventing proposals that violate the structural constraints of the model space.

Canonical form. Models are hashed after relabelling parameters by first appearance in a row-major scan and after taking the lexicographic minimum over permutations of the hidden classes, so label-switched duplicates of the same model are visited only once.

Saturated bound. When `lambda = 0` and `criterion` is `"AIC"` or `"BIC"`, a proposal is skipped without fitting whenever the log likelihood of the fully parameterized model cannot make it beat the incumbent.

Rejected proposals. Every proposal is put through a single coherence check that returns the reason it is not a model (an isolated state, multiple source components, an empty rate class, or no free parameters left), and the rejections are tallied by move type. The tally is printed with the object and is stored in each element of `attr(x, "dredge_history")`.

Stopping. A search ends either at `max.iterations` or when repeated deterministic polishing and reheating finds no improvement. The polisher evaluates valid one-parameter drops, eligible merges, frees, hidden-class collapses, and relocations of floor-supported transitions. Which condition ended each rate-category search is stored in the search history.

Score cache. Fitted models are cached by canonical form, so a structure is never fit twice. The chain is still free to revisit one: a cached proposal goes through the usual acceptance test using its stored fit.

Acceptance criterion. A proposed model within two units of the current score is always accepted, since models that close are indistinguishable and the chain should be free to cross the plateau. A worse model may still be accepted with probability $\exp(-(\Delta - 2)/T)$, where Δ is the score

difference and T is the current temperature. As temperature cools the algorithm transitions from broad exploration to focused exploitation.

Restart mechanism. To avoid entrapment in local optima the algorithm periodically restarts from a previously accepted model when the search has stagnated (no improvement in `restart.interval` iterations). The restart event is recorded in the trace for diagnostic purposes.

Regularization. An optional L1 or L2 penalty on the rate parameters is incorporated directly into the likelihood during optimisation. The `lambda` parameter controls the penalty strength. Setting `lambda = 0` recovers standard maximum likelihood, making scores directly comparable with `corHMM`.

Value

An object of class `"corhmm.dredge"`, which is a list of unique accepted models pruned to remove redundant entries (identical likelihoods and parameter counts). Each element is a standard `corhmm` object including `$phy`, `$data`, and `$data.legend`. The list is ordered by the information criterion score, best model first.

Full simulated annealing diagnostics for each rate category explored (per-iteration scores, move types, acceptance decisions) are stored as a hidden attribute and can be accessed via `attr(result, "dredge_history")`. This attribute is used automatically by `plotDredgeTrace` and is displayed in the `print` summary.

Author(s)

James D. Boyko

References

Boyko, J.D. (2026). Automatic Discovery of Optimal Discrete Character Models. Systematic Biology, In press.

Lewis, P.O. (2001). A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology, 50(6), 913-925.

Pagel, M., & Meade, A. (2006). Bayesian analysis of correlated evolution of discrete characters by reversible-jump Markov chain Monte Carlo. The American Naturalist, 167(6), 808-825.

Kalyaanamoorthy, S., Minh, B.Q., Wong, T.K.F., von Haeseler, A., & Jermiin, L.S. (2017). ModelFinder: fast model selection for accurate phylogenetic estimates. Nature Methods, 14, 587-589.

Burnham, K.P., & Anderson, D.R. (2002). Model selection and multimodel inference: a practical information-theoretic approach. Springer-Verlag.

Zhou, Y., Gao, M., Chen, Y., Shi, X. (2024). Adaptive Penalized Likelihood method for Markov Chains.

See Also

`corHMM`, `plotDredgeTrace`, `kFoldCrossValidation`, `getModelTable`

Examples

```
## Not run:
library(corHMM)
data(primates)
phy <- multi2di(primates[[1]])
phy$edge.length <- phy$edge.length + 1e-6
data <- primates[[2]]

# basic dredge search -- single rate category, L1 penalty
dredge_fits <- corHMMDredge(phy = phy,
  data = data,
  max.rate.cat = 1,
  pen.type = "l1",
  root.p = "maddfitz",
  lambda = 1)

# printing shows a model table and the best model
print(dredge_fits)

# trace plot uses the hidden dredge_history attribute automatically
plotDredgeTrace(dredge_fits)

# inspect the model table directly
mod_table <- getModelTable(dredge_fits)
print(mod_table)

# the best model for downstream analysis
best_fit <- dredge_fits[[which.min(mod_table$dAIC)]]

# access full SA diagnostics if needed
history <- attr(dredge_fits, "dredge_history")

# note: models inside dredge_history are stripped of $phy, $data, and
# $data.legend to save memory during the search. if you need complete
# objects, reattach the shared fields manually:
for(i in seq_along(history)){
  for(j in seq_along(history[[i]]$accepted_models)){
    history[[i]]$accepted_models[[j]]$phy         <- dredge_fits[[1]]$phy
    history[[i]]$accepted_models[[j]]$data        <- dredge_fits[[1]]$data
    history[[i]]$accepted_models[[j]]$data.legend <- dredge_fits[[1]]$data.legend
  }
}

for(i in seq_along(history)){
  for(j in seq_along(history[[i]]$every_model)){
    if(is.null(history[[i]]$every_model[[j]])){next}
    history[[i]]$every_model[[j]]$phy         <- dredge_fits[[1]]$phy
    history[[i]]$every_model[[j]]$data        <- dredge_fits[[1]]$data
    history[[i]]$every_model[[j]]$data.legend <- dredge_fits[[1]]$data.legend
  }
}
```

```
# resume an interrupted run using checkpointing
# dredge_fits <- corHMMDredge(phy = phy,
#   data = data,
#   max.rate.cat = 1,
#   lambda = 1,
#   checkpoint.file = "dredge_checkpoint.rds",
#   checkpoint.interval = 25)

## End(Not run)
```

---

corHMMDredgeBase *Fit a Single Penalized Hidden Markov Model*

Description

corHMMDredgeBase fits a single hidden Markov model (HMM) to a phylogenetic tree and discrete character data with optional regularization. It is the workhorse called repeatedly by corHMMDredge during the simulated annealing search, but can also be called directly to fit a specific model structure.

Usage

```
corHMMDredgeBase(phy, data, rate.cat, root.p="maddfitz", tip.fog=NULL,
  fog.ip=0.01, pen.type="l1", lambda=1, rate.mat=NULL,
  node.states="marginal", fixed.nodes=FALSE, ip=NULL, nstarts=0,
  n.cores=1, get.tip.states=FALSE, lewis.asc.bias=FALSE, collapse=FALSE,
  lower.bound=1e-10, upper.bound=100, opts=NULL, p=NULL,
  use_RTMB=FALSE, prep=NULL)
```

Arguments

phy	A phylogenetic tree of class "phylo".
data	A data frame with species names in the first column and discrete character states in subsequent columns.
rate.cat	An integer specifying the number of hidden rate categories.
root.p	Root state prior. Can be "maddfitz", "yang", a numeric vector, or NULL. See corHMM.
tip.fog	Optional tip observation error. A fixed numeric value less than 1 is treated as a per-state misclassification probability; if sum(tip.fog) >= 1 the probabilities are estimated.
fog.ip	Starting value for tip fog estimation. Defaults to 0.01.
pen.type	Regularization type: "l1" (lasso), "l2" (ridge), "er" (equal-rates penalty), or "unreg" (none). Defaults to "l1".
lambda	Penalty strength from 0 (no penalty) to 1 (full penalization). Defaults to 1.
rate.mat	An optional index matrix specifying the parameter structure of the rate matrix. If NULL an all-rates-different model is used.

node.states	Ancestral state reconstruction method: "marginal", "joint", "scaled", or "none".
fixed.nodes	Logical. Whether internal node states are fixed via node labels in phy.
ip	Optional starting values for optimisation.
nstarts	Number of random restarts. Defaults to 0.
n.cores	Number of cores for parallel random restarts.
get.tip.states	Logical. Whether to return tip state reconstructions.
lewis.asc.bias	Logical. Whether to correct for ascertainment bias following Lewis (2001).
collapse	Logical. Whether to collapse unobserved state combinations.
lower.bound, upper.bound	Bounds for rate parameter estimates.
opts	A list of options passed to the nloptr optimiser.
p	Optional fixed parameter vector. If supplied the likelihood is evaluated at these values without optimisation.
use_RTMB	Logical. Whether to use the RTMB automatic differentiation backend for gradient-based optimisation. Defaults to FALSE when called directly; corHMMDredge sets this to TRUE by default.
prep	An optional precomputed prep object from corHMMDredgePrep. When supplied, expensive data processing steps are skipped. This is used internally by corHMMDredge to avoid redundant computation across many model fits within the same rate category.

Details

There are three penalty types available:

"l1" penalizes the likelihood by the mean transition rate (analogous to lasso regression). The mean rather than sum is used to avoid over-penalizing complex models simply because they have more parameters.

"l2" penalizes by the squared mean transition rate (analogous to ridge regression). This is the most aggressive penalty, shrinking parameters most rapidly.

"er" penalizes by the average distance between parameter values. An equal-rates model incurs no penalty; models with one outlier rate incur a penalty unless strongly supported by the data. In practice this behaves similarly to an unpenalized model.

Setting pen.type = "unreg" or lambda = 0 recovers the standard unpenalized likelihood, identical to corHMM.

When use_RTMB = TRUE the gradient is computed analytically via automatic differentiation, which substantially improves optimisation speed and stability for models with many parameters.

Many arguments are shared with corHMM; see that documentation for further detail.

Value

An object of class "corhmm" with elements:

$loglik	Maximum log-likelihood.

`$AIC`	Akaike information criterion.
`$AICc`	AIC corrected for small sample size.
`$BIC`	Bayesian information criterion using the number of tips as the sample size.
`$rate.cat`	Number of rate categories.
`$solution`	Matrix of maximum likelihood rate estimates.
`$index.mat`	Index matrix of estimated parameters.
`$data`	Input dataset.
`$data.legend`	Input dataset with corHMM trait coding appended.
`$phy`	Input phylogeny.
`$states`	Most likely states at internal nodes.
`$tip.states`	Most likely states at tips (if requested).
`$states.info`	Information content (bits) at each node.
`$iterations`	Number of optimiser iterations.
`$root.p`	Root prior used in estimation.
`$tip.fog.p`	Estimated or fixed tip fog probabilities (if `tip.fog` was specified).

Author(s)

James D. Boyko

References

Boyko, J.D. (2026). Automatic Discovery of Optimal Discrete Character Models. Systematic Biology, In press.

Lewis, P.O. (2001). A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology, 50(6), 913-925.

Zhou, Y., Gao, M., Chen, Y., Shi, X. (2024). Adaptive Penalized Likelihood method for Markov Chains.

See Also

`corHMMDredge`, `corHMM`

Examples

```
library(corHMM)
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]

# fit a single model with L1 regularization
fit <- corHMMDredgeBase(phy = phy, data = data, rate.cat = 1,
  pen.type = "l1", root.p = "maddfitz", lambda = 1)
fit
```

```
# fit a specific model structure (note that dredge doesn't allow for collapsing)
index_mat <- getStateMat4Dat(data, collapse = FALSE)$rate.mat
fit2 <- corHMMDredgeBase(phy = phy, data = data, rate.cat = 1,
  rate.mat = index_mat, pen.type = "l1", lambda = 0)
fit2
```

`dent_likelihood` *Dents the likelihood surface This takes any values that are better (lower) than the desired negative log likelihood and reflects them across the best_neglnL + delta line, "denting" the likelihood surface.*

Description

Dents the likelihood surface This takes any values that are better (lower) than the desired negative log likelihood and reflects them across the best_neglnL + delta line, "denting" the likelihood surface.

Usage

```
dent_likelihood(neglnL, best_neglnL, delta = 2)
```

Arguments

`neglnL`	The original negative log likelihood
`best_neglnL`	The negative log likelihood at the optimum; other values will be greater than this.
`delta`	How far from the optimal negative log likelihood to focus samples

Value

The transformed negative log likelihood

`dent_propose` *Propose new values This proposes new values using a normal distribution centered on the original parameter values, with desired standard deviation. If any proposed values are outside the bounds, it will propose again.*

Description

Propose new values This proposes new values using a normal distribution centered on the original parameter values, with desired standard deviation. If any proposed values are outside the bounds, it will propose again.

Usage

```
dent_propose(old_params, lower_bound = -Inf, upper_bound = Inf, sd = 1)
```

Arguments

old_params	The original parameter values
lower_bound	Minimum parameter values to try. One for all or a vector of the length of par.
upper_bound	Maximum parameter values to try. One for all or a vector of the length of par.
sd	Standard deviation to use for the proposals. One for all or a vector of the length of par.

Value

A vector of the new parameter values

---

dent_walk

Sample points from along a ridge This "dents" the likelihood surface by reflecting points better than a threshold back across the threshold (think of taking a hollow plastic model of a mountain and punching the top so it's a volcano). It then uses essentially a Metropolis-Hastings walk to wander around the new rim. It adjusts the proposal width so that it samples points around the desired likelihood. This is better than using the curvature at the maximum likelihood estimate since it can actually sample points in case the assumptions of the curvature method do not hold. It is better than varying one parameter at a time while holding others constant because that could miss ridges: if I am fitting 5=x+y, and get a point estimate of (3,2), the reality is that there are an infinite range of values of x and y that will sum to 5, but if I hold x constant it looks like y is estimated very precisely. Of course, one could just fully embrace the Metropolis-Hastings lifestyle and use a full Bayesian approach.

Description

While running, it will display the current range of likelihoods in the desired range (by default, the best negative log likelihood + 2 negative log likelihood units) and the parameter values falling in that range. If things are working well, the range of values will stabilize during a search.

Usage

```
dent_walk(
  par,
  fn,
  best_neglnL,
  delta = 2,
  nsteps = 1000,
```

```
  print_freq = 50,
  lower_bound = 0,
  upper_bound = Inf,
  adjust_width_interval = 100,
  badval = 1e+09,
  sd_vector = NULL,
  debug = FALSE,
  restart_after = 50,
  ...
)
```

Arguments

`par`	Starting parameter vector, generally at the optimum. If named, the vector names are used to label output parameters.
`fn`	The likelihood function, assumed to return negative log likelihoods
`best_neglnL`	The negative log likelihood at the optimum; other values will be greater than this.
`delta`	How far from the optimal negative log likelihood to focus samples
`nsteps`	How many steps to take in the analysis
`print_freq`	Output progress every print_freq steps.
`lower_bound`	Minimum parameter values to try. One for all or a vector of the length of par.
`upper_bound`	Maximum parameter values to try. One for all or a vector of the length of par.
`adjust_width_interval`	When to try automatically adjusting proposal widths
`badval`	Bad negative log likelihood to return if a non-finite likelihood is returned
`sd_vector`	Vector of the standard deviations to use for proposals. Generated automatically if NULL
`debug`	If TRUE, prints out much more information during a run
`restart_after`	Sometimes the search can get stuck outside the good region but still accept moves. After this many steps without being inside the good region, restart from one of the past good points
`...`	Other arguments to fn.

Details

The algorithm tunes: if it is moving too far away from the desired likelihoods, it will decrease the proposal width; if it staying in areas better than the desired likelihood, it will increase the proposal width. It will also expand the proposal width for parameters where the extreme values still appear good enough to try to find out the full range for these values.

In general, the idea of this is not to give you a pleasingly narrow range of possible values – it is to try to find the actual uncertainty, including finding any ridges that would not be seen in univariate space.

Value

A dentist object containing results, the data.frame of negative log likelihoods and the parameters associated with them; acceptances, the vector of whether a proposed move was accepted each step; best_neglnL, the best value passed into the analysis; delta, the desired offset; all_ranges, a summary of the results.

---

examples *Example datasets*

Description

Example files for running various functions in `corHMM`. The "primates" dataset comes from the example files provided by BayesTraits, though here we only include a single tree with branch lengths scaled to time. The "primates.paint" dataset is the same, but with the tree painted according to hypothetical regimes. Finally, the "rayDISC.example" dataset provides an example on how polymorphic data can be coded for `rayDISC`.

Format

a list object that contains a tree of class "phylo" and a dataframe that contains the trait data

References

Pagel, M., and A. Meade. 2006. Bayesian analysis of correlated evolution of discrete characters by reversible-jump Markov chain Monte Carlo. American Naturalist 167:808-825.

---

fitCorrelationTest *Test for correlation*

Description

Automatically fits a set of independent and dependent models to test for correlation between characters.

Usage

```
fitCorrelationTest(phy,
  data,
  root.p="yang",
  nstarts=0,
  n.cores=1,
  simplified_models=FALSE,
  use_RTMB=TRUE)
```

Arguments

- `phy` — a phylogenetic tree, in `ape` "phylo" format.
- `data` — a data.frame containing species information. The first column must be species names matching the phylogeny. Additional columns contain discrete character data.
- `root.p` — The root prior being used. Default is "yang", but see corHMM documentation for more details.
- `nstarts` — the number of random restarts to be performed. The default is `nstarts=0`.
- `n.cores` — the number of processor cores to spread out the random restarts.
- `simplified_models` — A boolean which indicates whether to include simplified independent and dependent models (currently only works for two binary-state characters; see Details).
- `use_RTMB` — A boolean indicating whether to use autodifferentiation (much faster).

Details

This function automatically fit a set of multi-rate independent and dependent models (with default corHMM options) to drastically reduce false support for correlation. Currently, the simplified models are only available for two binary-state characters, but it is straightforward for users to use the tools available in corHMM to create model structures specific to their questions when the datasets are more complex.

The correlation between two characters is often interpreted as evidence that there exists a significant and biologically important relationship between them. However, Maddison and FitzJohn (2015) recently pointed out that in certain situations find evidence of correlated evolution between two categorical characters is often spurious, particularly, when the dependent relationship stems from a single replicate deep in time. In Boyko and Beaulieu (in prep) we showed that there is, in fact, a statistical solution to the problem posed by Maddison and FitzJohn (2015) naturally embedded within the expanded model space afforded by the hidden Markov model (HMM) framework.

The problem of single unreplicated evolutionary events manifests itself as rate heterogeneity within our models and that this is the source of the false correlation. Therefore, we argue that this problem is better understood as model misspecification rather than a failure of comparative methods to account for phylogenetic pseudoreplication. We utilize HMMs to develop a multi-rate independent model which, when implemented, drastically reduces support for correlation.

Value

`fitCorrelationTest` returns an object of class `corhmm_list`. This is a list with elements:

- `$independent_model_fit` — A corHMM object of the standard independent model ala Pagel (1994).
- `$correlated_model_fit` — A corHMM object of the standard dependent model ala Pagel (1994).
- `$hidden_Markov_independent_model_fit` — A corHMM object of the hidden Markov independent model which allows for rate heterogeneity independent of the focal character.

`$hidden_Markov_correlated_model_fit.cat`
: A corHMM object of the hidden Markov dependent model which allows for rate heterogeneity independent of the focal character as well as correlation between characters.

`$simplified_*`
: If simplified was set to TRUE, then the function will also return simplified versions of the above models. These models have fewer parameters than the above models while still being either dependent or independent models.

Author(s)

James D. Boyko

References

Maddison W.P., FitzJohn R.G. 2015. The Unsolved Challenge to Phylogenetic Correlation Tests for Categorical Characters. Systematic Biology. 64:127-136. Boyko J.D., Beaulieu J.M. 2023. Reducing the Biases in False Correlations Between Discrete Characters. Systematic Biology. 72:476-488.

Examples

```
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
# not run because of long run times
#corr_test_fits <- fitCorrelationTest(phy = phy, data = data, simplified_models = TRUE)
#corr_test_fits
```

`get_batch_profile_lik` *Perform Batch Profile Likelihood Analysis for Multiple Parameters*

Description

`get_batch_profile_lik` is a wrapper function that performs profile likelihood analysis for multiple parameters of a fitted `corHMM` object. It allows exploration of the likelihood surface by evaluating the likelihood at various points along the parameter space.

Usage

```
get_batch_profile_lik(corhmm_obj,
  range_factor,
  n_points,
  verbose=FALSE,
  ncores=NULL,
  dredge=FALSE)
```

Arguments

corhmm_obj	A fitted corHMM object representing the model for which profile likelihood analysis is to be performed.
range_factor	A numeric factor determining the range over which to generate points for the profile likelihood analysis. This value is used to calculate the bounds around the maximum likelihood estimates (MLEs) for each parameter.
n_points	The number of points to generate for each parameter along its profile likelihood curve.
verbose	Whether to print messages about which parameter is being optimized.
ncores	The number of processor cores to be used for parallel computation. Default is NULL, which uses a single core.
dredge	Logical value indicating whether to include model penalization factors such as pen_type and lambda from the fitted corHMM object. Default is FALSE.

Details

This function performs a profile likelihood analysis for each parameter in a fitted corHMM model. It evaluates the likelihood at logarithmically spaced points around the maximum likelihood estimates (MLEs) for the parameters. If dredge is set to TRUE, the function also considers model penalization terms.

The function works by first generating a set of points along a logarithmic scale for each parameter, then fixing one parameter at a time while optimizing over the others. The resulting profile likelihood values are returned for each parameter.

Parallel computation can be enabled using the ncores argument to speed up the analysis.

Value

A list containing the profile likelihood results for each parameter. Each entry in the list corresponds to a parameter and contains the profile likelihood values across the range of points evaluated. The original corHMM object is also included in the output.

Author(s)

Your Name

See Also

corHMM for fitting hidden Markov models to phylogenetic data.

Examples

```
# Assuming you have a fitted corHMM object:
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
corhmm_fit <- corHMM(phy = phy, data = data, rate.cat = 1)
profile_results <- get_batch_profile_lik(corhmm_fit, range_factor = 2, n_points = 50)
```

```
plot_batch_profile_lik(profile_results)
```

`getCVTable` *Print Method for corhmm.kfold Class Objects*

Description

Summarize the scores for each fold and the average cross-validation score for objects of class `"corhmm.kfold"`.

Usage

```
getCVTable(x)
```

Arguments

`x`	An object of class `"corhmm.kfold"` returned from `kFoldCrossValidation`.

Value

Returns a list of scores per fold for all lambdas and an average score for each lambda.

Author(s)

James D. Boyko

`getFullMat` *Combines several rate class index matrices*

Description

Combines several index matrices which describe transitions between observed states into output a single index matrix for use in corHMM

Usage

```
getFullMat(StateMats, RateClassMat = NULL)
```

Arguments

`StateMats`	A list of index matrices describing transitions between observed states. Each unique number from 1 to n, will be independently estimated. Values of 0 are not estimated. Matrix entries of the same value are estimated to be the same rate.
`RateClassMat`	An optional index matrix which describes how StateMats are related to one another. This will be a matrix of size: length(StateMats) by length(StateMats). By default, all transitions between StateMats are allowed and independently estimated.

Details

This function is the final step in creating a custom hidden Markov model. It takes a list of index matrices (StateMats) which describe different ways that the observed states are related to one another and creates a single matrix to describe the model. The matrices are combined following Eq. 2 of Tarasov (2019). getFullMat is part of several functions which help the user efficiently create custom index matrices. Often, it will be more practical to begin constructing a custom model with `getRateMat4Dat`.

`getStateMat` will generate an index matrix of size n by n in which all transitions between the n states are allowed and independently estimated. That index matrix can then be manipulated by `dropStateMatPars` and `equateStateMatPars`. `dropStateMatPars` will drop specific rates from an index matrix. `dropStateMatPars` requires an index matrix and a vector of which rates should be dropped. `equateStateMatPars` will equate rates within an index matrix. `equateStateMatPars` requires an index matrix and a list of vectors, each element of which should correspond to two or more rates to be equated.

Value

Returns an index matrix.

Author(s)

James D. Boyko

References

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Tarasov, S. 2019. Integration of Anatomy Ontologies and Evo-Devo Using Structured Markov Models Suggests a New Framework for Modeling Discrete Phenotypic Traits. Systematic Biology, 68(5) 698-716. doi:10.1093/sysbio/syz005

See Also

`getRateMat4Dat`

Examples

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
# create a legend and rate mat from a multi-character dataset.
LegendAndRateMat <- getStateMat4Dat(data)
rate.mat <- LegendAndRateMat$rate.mat
legend <- LegendAndRateMat$legend

# To create a hidden markov model first define your rate classes (state-dependent processes)
# R1 will be a manually created SYM model
R1 <- equateStateMatPars(rate.mat, c(1:6))
```

```
# R2 will only allow transitions between 1 and 2
R2 <- dropStateMatPars(rate.mat, c(3,4))
# R1 and R2 will transtion at equal rates (i.e. the parameter process will be ER)
P <- getRateCatMat(2)
P <- equateStateMatPars(P, c(1,2))
# combine our state-dependnet processes and parameter process
HMM <- getFullMat(list(R1, R2), P)

# This can now be used in a corHMM model
CustomModel <- corHMM(phy = phy, data = data, rate.cat = 2, rate.mat = HMM, node.states = "none")
```

`getModelTable` *Summarize Model Statistics for a List of corHMM Objects*

Description

`getModelTable` extracts key statistics from a list of fitted `corHMM` models and returns a summary table. The table includes the number of parameters, log-likelihood, and model selection criteria such as AIC, delta AIC, and AIC weights.

Usage

```
getModelTable(model_list, type = "AIC")
```

Arguments

`model_list`	A list of `corHMM` objects, each representing a fitted hidden Markov model to phylogenetic and character data.
`type`	The type of model selection criterion to use. Options are `"AIC"`, `"AICc"`, or `"BIC"`. Default is `"AIC"`.

Details

This function takes a list of models fitted using `corHMM` and calculates key statistics for comparison across models. Specifically, it calculates the number of parameters, log-likelihood, the chosen model selection criterion (e.g., AIC), the difference in the criterion relative to the best model (delta AIC), and the relative model weight based on the criterion.

`getModelTable` can handle different model selection criteria such as AIC, AICc, and BIC by specifying the `type` argument.

Value

A data frame with the following columns:

- `np`: The number of parameters in the model.
- `lnLik`: The log-likelihood of the model.
- `AIC` (or the value of `type`): The model selection criterion value.

- dAIC: The difference in the criterion between the model and the best model (i.e., the model with the minimum criterion value).
- AICwt (or the corresponding criterion-weight column): Relative model weights calculated from criterion differences.

Author(s)

James D. Boyko

See Also

corHMM for fitting hidden Markov models to phylogenetic data.

Examples

```
# Assuming you have a list of fitted corHMM models:
#models <- list(model1, model2, model3)
#getModelTable(models)
# To use BIC instead of AIC:
#getModelTable(models, type = "BIC")
```

---

getStateMat4Dat *Produce an index matrix and legend from a dataset*

Description

Takes a dataset to produce an index matrix that corresponds to a single state-dependent process (i.e. a single rate category) and a legend which matches input data to the rows and columns of the index matrix and corHMM solution.

Usage

```
getStateMat4Dat(data, model = "ARD", dual = FALSE, collapse = TRUE, indep = FALSE,
CorData = NULL)
```

Arguments

data	A data matrix containing species information in the same format as the main corHMM function: column[,1] is species names, column[,2:n] are the discrete states.
model	One of "ARD", "SYM", or "ER". ARD: all rates differ. SYM: rates between any two states do not differ. ER: all rates are equal.
dual	A boolean indicating whether or not to include dual transitions.

`collapse` a boolean indicating whether to collapse multiple character combinations into only the observed states. For example, if true a two character dataset contained (0,0), (1,0), and (1,1), this would be collapsed into 1,2,3. However, if set to false it would 1,2,4. In combination with a custom rate matrix this allows for the estimation of transitions between the unobserved character combination. The default is `TRUE`

.

`indep` A boolean indicating whether or not to return an independent or correlated model.

`CorData` An optional pre-processed data object as returned by `corProcessData`. Supplying it skips the internal call to `corProcessData`, which is useful when building several rate matrices from the same dataset. The default, `NULL`, processes `data` internally.

Details

This function will generate an index matrix based on user provided data. It provides a useful starting point for further modifications using `dropStateMatPars`, `equateStateMatPars`, and `getFullMat`. If more than a single column of data is given double transitions between characters are disallowed. For example, if character 1 is the presence or absence of limbs, and character 2 is the presence or absence of fingers, then the transition from absence of limbs and fingers to presence of limbs and fingers is automatically disallowed. This is consistent with Pagel's (1994) model of correlated character evolution.

Value

`$legend` A named vector. The elements of the vector are all the unique state combinations in the user data. The names of the vector are the state number assigned to each combination.

`$rate.mat` A rate index matrix describing a single rate class.

Author(s)

James D. Boyko

References

Pagel, M. 1994. Detecting correlated evolution on phylogenies: a gneeral method for the comparative analysis of discrete characters. Proc. R. Soc. Lond. B 255:37-45.

See Also

`getFullmat`

Examples

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
# create a legend and rate mat from a multi-character dataset.
LegendAndRateMat <- getStateMat4Dat(data)
rate.mat <- LegendAndRateMat$rate.mat
legend <- LegendAndRateMat$legend
```

---

kFoldCrossValidation *Perform K-Fold Cross-Validation for corHMM Models*

Description

This function performs k-fold cross-validation on a given `corHMM` model by dividing the data into k equally sized subsets. The function evaluates model performance across multiple lambda regularization values, if provided. Optionally, it can save the trained models for each fold and return the cross-validation results.

Usage

```
kFoldCrossValidation(corhmm_obj, k, lambdas = NULL, return_model = TRUE,
save_model_dir = NULL, model_name = NULL)
```

Arguments

`corhmm_obj`	A `corHMM` object that contains a fitted model.
`k`	An integer specifying the number of folds to divide the data into for cross-validation.
`lambdas`	A numeric vector of lambda regularization values to evaluate during cross-validation. If `NULL`, the lambda value from `corhmm_obj` will be used. Defaults to `NULL`.
`return_model`	A logical value indicating whether to return the trained models for each fold. Defaults to `TRUE`.
`save_model_dir`	A character string specifying the directory to save the trained models for each fold. If `NULL`, models will not be saved. Defaults to `NULL`.
`model_name`	A character string specifying the base name for saved model files. If `NULL`, a default name `"corhmm.obj"` is used. Defaults to `NULL`.

Details

The function splits the data into `k` folds and trains a separate `corHMM` model for each fold by leaving one fold out as the test set. The remaining folds are used for training the model. The performance of the model is evaluated on the test set using a divergence-based (Jensen-Shannon Divergence)

scoring method. Evaluations are based on estimating the tips which were removed for that particular fold given the newly fitted model.

The function supports evaluating models across different lambda regularization values. If `lambdas` are provided, models are trained and evaluated for each lambda value. The results, including the models (if `return_model = TRUE`) and cross-validation scores, are returned as a list.

Value

A list of cross-validation results, including the following components:

`models`	A list of the trained models for each fold (if `return_model = TRUE`).
`scores`	A numeric vector of the cross-validation scores for each fold.
`averageScore`	The average cross-validation score across all folds.

Author(s)

James D. Boyko

Examples

```
## Not run:
# run your dredge
data(primates)
phy <- multi2di(primates[[1]])
data <- primates[[2]]
dredge_fits <- corHMMDredge(phy = phy, data = data,
  max.rate.cat = 1, pen.type = "l1",
  root.p = "maddfitz", lambda = 1, nstarts = 10, n.cores = 10)

# select the best model and run CV
model_table <- getModelTable(dredge_fits)
dredge_model <- dredge_fits[[which.min(model_table$dAIC)]]
k_fold_res <- kFoldCrossValidation(dredge_model,
k = 5, lambdas = c(0,0.25,0.5,0.75,1))

cv_table <- getCVTable(k_fold_res)

# choose lowest average score for lambda
print(cv_table)

## End(Not run)
```

`makeSimmap` *Simulate a character history*

Description

Produces a character history from a fitted corHMM object or a transition rate matrix.

Usage

```
makeSimmap(tree, data, model, rate.cat, root.p="yang", nSim=1, nCores=1, fix.node=NULL,
fix.state=NULL, parsimony = FALSE, max.attempt = 1e5, collapse=TRUE,
method=c("rejection", "uniformization"))
```

Arguments

`tree` A phylogeny of class phylo. If omitted when `model` is a fitted corHMM object, uses the fit's phylogeny.

`data` a data.frame containing species information. The first column must be species names matching the phylogeny. Additional columns contain discrete character data. Omit this argument when `model` is a fitted corHMM object; its saved `data.legend` supplies the tip-state indices.

`model` A transition rate matrix or a fitted object of class `corhmm`. Passing the complete fit preserves the state coding used during fitting, including for fits from older package versions.

`rate.cat` The number of rate categories. Taken from `model` when it is a fitted corHMM object; an explicitly supplied value must agree with the fit.

`root.p` The root prior to begin the sampling at the root. If omitted when `model` is a fitted corHMM object, uses the fit's root prior.

`nSim` The number of simmaps to be simulated.

`nCores` The number of cores to be used.

`fix.node` A vector specifying node numbers to be fixed. Also possible to fix tips if using a hidden Markov model. Tips are in the order of tree$tip.label.

`fix.state` Specifies which states to fix the nodes. States are specified according to position in the rate matrix. E.g. If I had binary observed characters 0/1 and two hidden rate classes A/B and wanted to fix a node as 1B, I would set this to 4.

`parsimony` A boolean indicating whether node states should be based on conditional likelihood (per Bollback 2006), or if they should be consistent with a parsimonious model (if TRUE). Parsimony states are evaluted by dividing the rates present in the variable, `model`, by 1000 and evaluating the conditional likelihood of each state. However, by lowering the rates we can approximate a parsimony reconstruction (Steel and Penny 2000).

`max.attempt` For `method = "rejection"`, the maximum number of attempts to create a path between the states at the ends of a branch. When this value is reached, the Floyd-Walsh algorithm produces a shortest path with equally spaced changes. Ignored for `method = "uniformization"`.

`collapse` a boolean indicating whether to collapse multiple character combinations into only the observed states. For example, if true a two character dataset contained (0,0), (1,0), and (1,1), this would be collapsed into 1,2,3. However, if set to false it would 1,2,4. In combination with a custom rate matrix this allows for the estimation of transitions between the unobserved character combination. The default is `TRUE`. When `model` is a fitted corHMM object, this argument is not used: the fitted state space and saved indices are retained.

`method` The within-branch history sampler. `"rejection"` retains the existing behavior. `"uniformization"` samples paths conditional on their endpoints using a Poisson process and a discrete Markov chain, with shared precomputed matrix powers. It avoids rejected paths and the shortest-path fallback, and can be faster when the required endpoint is unlikely.

Details

This function will generate a character history given a model and dataset. It has a similar structure to the simmap generated in phytools and follows the methods of Bollback (2006). Conditional tip and node likelihoods are computed during mapping; saved ancestral reconstructions are not required. We chose not to implement any new plotting functions, instead `makeSimmap` produces a simmap object which is formatted so it can used with other R packages such as phytools (Revell, 2012). For additional capabilities, options, and biological examples we refer readers to the detailed _Generalized corHMM_ vignette.

For mapping an existing fit, use `makeSimmap(model = fit, nSim = 100)`. The fit must retain its original `solution`, `rate.cat`, and `data.legend`. Numeric root priors and `fix.state` continue to refer to positions in the fitted matrix. Matrix state names are retained in the maps.

When passing a matrix instead, its rows and columns must be in the same state order as the supplied data are assigned by the installed version of corHMM. Column names label the output; they do not align tip states to matrix positions. Version 2.8 sorted combined character states, while 2.10.x uses the order from `expand.grid`. For two binary characters with all four combinations, these orders are 00, 01, 10, 11 and 00, 10, 01, 11, respectively. Setting `collapse = TRUE` selects the observed combinations but does not restore the older ordering. Passing the complete fit avoids this mismatch without refitting. Maps previously generated with mismatched state coding should be regenerated; changing their labels alone does not correct the histories.

Uniformization follows Hobolth and Stone (2009). Virtual self-transitions are removed from the returned histories. The Poisson expansion is truncated with omitted conditional probability bounded by 10^{-12}, using the computed branch transition probabilities. Tables exceeding 10,000 virtual events or 20 million entries produce an error suggesting the rejection method. The two methods sample the same target distribution before the rejection method's fallback is reached; speed depends on the rates, branch lengths, and endpoints.

Value

A list of simmaps.

Author(s)

James D. Boyko

References

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Bollback, J. P. 2006. SIMMAP: stochastic character mapping of discrete traits on phylogenies. BMC Bioinformatics 7:88.

Hobolth, A., and E. A. Stone. 2009. Simulation from endpoint-conditioned, continuous-time Markov chains on a finite state space, with applications to molecular evolution. Annals of Applied Statistics 3:1204-1231.

Revell, L. J. 2012. phytools: an R package for phylogenetic comparative biology (and other things). Methods in Ecology and Evolution, 3(2), 217-223.

Steel, M., and D. Penny. 2000. Parsimony, Likelihood, and the Role of Models in Molecular Phylogenetics. Molecular Biology and Evolution 17:839-850.

Examples

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]

##run corhmm
MK <- corHMM(phy, data, 1)

## preserve the fitted state coding, including across package versions
simmaps <- makeSimmap(model=MK, nSim=100, nCores=1)
# To use uniformization instead, add method="uniformization".

## get the summary
simmap_summaries <- lapply(simmaps, summarize_single_simmap)
summary_df <- summarize_transition_stats(simmap_summaries)
print(summary_df)

## we import phytools plotSimmap for plotting
# library(phytools)
# plotSimmap(simmap[[1]])
```

---

node_importance_by_cluster

Identify nodes that distinguish clusters of reconstructions

Description

For every internal node and every cluster of sampled joint reconstructions, scores how diagnostic that node is of that cluster. High scores mark the nodes at which alternative evolutionary histories actually disagree, and are the nodes worth plotting and interpreting.

Usage

```
node_importance_by_cluster(state_df, clusters)
```

Arguments

state_df	Data frame of sampled reconstructions, rows being reconstructions and columns being internal nodes, as returned in state_df by compute_joint_ci.
clusters	Vector of cluster assignments, one per row of state_df.

Details

For a focal cluster and node, the modal state within the cluster is found. within_match is the proportion of reconstructions inside the cluster carrying that state and other_match is the proportion outside the cluster carrying it. Importance is within_match^2 * (1 - other_match), which is large only when a node is both consistent within the cluster and different from everything else.

Value

A data frame with one row per node per cluster and columns node, cluster, modal_state, within_match, other_match and importance.

Author(s)

James D. Boyko

See Also

plot_clustered_phylo

Examples

```
 ## Not run:
ni <- node_importance_by_cluster(ci$state_df, clusters)
subset(ni, importance > 0.5)

## End(Not run)
```

---

phylo_aware_dist_fast *Compile a faster version of the reconstruction distance*

Description

Compiles and returns a C++ implementation of phylo_aware_dist_local. corHMM ships no compiled code, so the C++ source is only built when you ask for it here. The result is numerically identical to the pure R version and roughly three times faster, which is worth the extra step only when you are clustering many thousands of reconstructions.

Usage

```
phylo_aware_dist_fast(verbose = FALSE)
```

Arguments

verbose	Logical. Pass TRUE to see the compiler output. Default FALSE.

Details

Requires the **Rcpp** package and a working C++ toolchain (Rtools on Windows, Xcode command line tools on macOS). Compilation takes a few seconds the first time in a session.

The returned function is only valid in the session that created it: it holds a pointer to code compiled into that session, so do not save it to an .rds or .RData file and reload it later. Call phylo_aware_dist_fast() again in each new session instead.

The C++ source is installed with the package and can be read or adapted from system.file("cpp", "phylo_aware_dist_local.cpp", package = "corHMM").

Value

A function with the same arguments and return value as phylo_aware_dist_local, suitable for registering with proxy::pr_DB.

Author(s)

James D. Boyko

See Also

phylo_aware_dist_local, compute_joint_ci

Examples

```
 ## Not run:
fast_dist <- phylo_aware_dist_fast()
proxy::pr_DB$set_entry(FUN = fast_dist, names = "PhyloSpreadDistFast")
D <- proxy::dist(as.matrix(ci$state_df), method = "PhyloSpreadDistFast",
phylo_dist = phylo_dist)

## End(Not run)
```

---

phylo_aware_dist_local

Phylogenetically aware distance between two reconstructions

Description

Computes a dissimilarity between two joint ancestral state reconstructions that accounts for where on the phylogeny they disagree. Reconstructions that differ at several phylogenetically adjacent nodes are treated as more dissimilar than reconstructions that differ at the same number of scattered nodes, so that the resulting distance matrix separates genuinely different evolutionary narratives rather than isolated node flips.

Usage

```
phylo_aware_dist_local(graph_labels_x, graph_labels_y, phylo_dist)
```

Arguments

`graph_labels_x`	Integer vector of internal node states for the first reconstruction.
`graph_labels_y`	Integer vector of internal node states for the second reconstruction, in the same node order.
`phylo_dist`	A square matrix of pairwise node distances among internal nodes, e.g. `ape::dist.nodes` applied to a tree with all branch lengths set to 1, subset to the internal nodes.

Details

Let V_{diff} be the set of internal nodes at which the two reconstructions differ. If $|V_{diff}| \leq 1$ the distance is 0. Otherwise the distance is $\sum \exp(1/d_P(v_a, v_b))$ summed over all unique pairs of nodes in V_{diff}, where d_P is the node distance supplied in `phylo_dist`.

The function is intended to be registered with the **proxy** package and used through `proxy::dist`, which supplies the pairwise arguments (see examples).

Value

A single non-negative numeric distance.

Author(s)

James D. Boyko

See Also

`compute_joint_ci`, `node_importance_by_cluster`

Examples

```
 ## Not run:
tmp <- phy; tmp$edge.length <- rep(1, nrow(tmp$edge))
phylo_dist <- ape::dist.nodes(tmp)[-(1:Ntip(tmp)), -(1:Ntip(tmp))]
proxy::pr_DB$set_entry(FUN = phylo_aware_dist_local, names = "PhyloSpreadDistLocal")
D <- proxy::dist(as.matrix(ci$state_df), method = "PhyloSpreadDistLocal",
phylo_dist = phylo_dist)

## End(Not run)
```

plot.dentist

Plot the dented samples This will show the univariate plots of the parameter values versus the likelihood as well as bivariate plots of pairs of parameters to look for ridges.

Description

Plot the dented samples This will show the univariate plots of the parameter values versus the likelihood as well as bivariate plots of pairs of parameters to look for ridges.

Usage

```
## S3 method for class 'dentist'
plot(x, ...)
```

Arguments

x	An object of class dentist
...	Other arguments to pass to plot

plot_batch_profile_lik

Plot Batch Profile Likelihoods

Description

Plots profile likelihoods for each parameter in a batch from the output of a profile corHMM object, displaying the log-likelihoods across parameter values and indicating the maximum likelihood estimate (MLE) and the 95% confidence interval.

Usage

```
plot_batch_profile_lik(corhmm_profile,
  n_cols = NULL,
  n_rows = NULL,
  mar = c(5, 4, 4, 1) + 0.1,
  ci_level = 1.96,
  polygon_col = "lightgrey",
  line_col = "black",
  line_type = "l",
  mle_col = "blue",
  ci_line_col = "black",
  ci_line_type = "dashed",
  axis_tick_length = -0.2,
  label_cex = 0.7,
```

```
  ylim=NULL,
  xlab="Parameter Value",
  ...)
```

Arguments

corhmm_profile	a list containing profile likelihood tables for each parameter and the corHMM object with MLE and loglik attributes.
n_cols	optional; number of columns for the plotting layout. If NULL, automatically calculated based on the number of parameters.
n_rows	optional; number of rows for the plotting layout. If NULL, automatically calculated based on the number of parameters.
mar	margins around the plot. Defaults to c(5, 4, 4, 1) + 0.1.
ci_level	z-value for the 95% confidence interval. Defaults to 1.96.
polygon_col	color of the polygon under the curve. Defaults to "lightgrey".
line_col	color of the profile likelihood curve. Defaults to "black".
line_type	type of the profile likelihood curve. Defaults to "l".
mle_col	color of the MLE point. Defaults to "blue".
ci_line_col	color of the 95% CI line. Defaults to "black".
ci_line_type	line type for the 95% CI line. Defaults to "dashed".
axis_tick_length	length of the axis ticks, with negative values indicating ticks pointing inwards. Defaults to -0.2.
label_cex	character expansion size for labels, affecting the size of text labels. Defaults to 0.7.
ylim	a user-specified upper and lower limit to the y-axis.
xlab	user-specified x label
...	Additional arugments to be passed to plot.

Details

This function generates a series of plots for the profile likelihood of each parameter in the input corHMM model object. It visualizes the log-likelihood across the range of parameter values, highlights the maximum likelihood estimate (MLE), and denotes the 95% confidence interval with a horizontal dashed line. The function is designed to accommodate the batch analysis of multiple parameters, organizing the plots in a specified layout and allowing for extensive customization of plot aesthetics.

Value

Generates and displays a grid of profile likelihood plots, one for each parameter in the input model, with visual cues for MLE and confidence intervals.

Author(s)

James D. Boyko

plot_clustered_phylo *Plot competing ancestral histories on a phylogeny*

Description

Draws the phylogeny with the best joint reconstruction at every internal node, and attaches to the diagnostic nodes of each cluster a pie chart of the states that cluster assigns there. This gives a single figure in which the alternative evolutionary histories implied by different clusters of reconstructions can be read off directly, rather than being blurred together as they would be in a marginal summary.

Usage

```
plot_clustered_phylo(phy, state_df, cluster_assignments, node_states,
important_nodes, cluster_colors = c("#E41A1C", "#377EB8", "#4DAF4A",
"#984EA3", "#FF7F00", "#FFFF33", "#A65628"),
state_colors = c("white", "black"), state_labels = NULL,
outer_cex = 2.2, inner_rad = 3, node_offset = 10, edge_width = 0.5,
show_legend = TRUE, legend_pos = "topleft", legend_title = NULL,
phylogram_params = list(), plot_params = list(),
point_params = list(pch = NULL, cex = NULL, col = NULL),
segment_params = list(lwd = NULL, lty = NULL), ...)
```

Arguments

`phy` A phylogeny of class `"phylo"`, matching the tree used to generate `state_df`.

`state_df` Data frame of sampled reconstructions from `compute_joint_ci`.

`cluster_assignments` Vector of cluster assignments, one per row of `state_df`.

`node_states` Integer vector of internal node states for the reconstruction drawn on the tree itself, typically `res$states` from the best joint reconstruction.

`important_nodes` A list with one element per cluster, in cluster order, each a data frame with a `node` column giving that cluster's diagnostic nodes. Usually built by thresholding `node_importance_by_cluster`.

`cluster_colors` Vector of colours for the clusters.

`state_colors` Vector of colours for the character states.

`state_labels` Optional character vector of state names for the legend.

`outer_cex` Size of the coloured ring drawn around each cluster pie.

`inner_rad` Radius of the pie chart drawn inside that ring.

`node_offset` Horizontal spacing used to fan out multiple clusters that flag the same node.

`edge_width` Width of the tree edges.

`show_legend`, `legend_pos`, `legend_title` Legend control.

`phylogram_params`, `plot_params`
: Named lists of additional arguments for `phylogram.plot` and `plot.phylo`.

`point_params`
: Named list of `pch`, `cex` and `col` for the base node points.

`segment_params`
: Named list of `lwd` and `lty` for the lines connecting cluster pies to the tree.

`...`
: Currently unused.

Details

Because clustering has already separated the sample into distinct global narratives, the pie charts show variation remaining *within* a cluster, not an average over all possible histories. Note that `important_nodes[[i]]` must correspond to cluster `i`; build the list by looping over cluster numbers rather than over `unique(clusters)`.

Value

Invisibly returns a list with the plotting coordinates `XX` and `YY`, the per-cluster point positions `point_list`, and the pie data `pie_dat`.

Author(s)

James D. Boyko

See Also

`compute_joint_ci`, `node_importance_by_cluster`, `plot_raw_recon`

Examples

```
 ## Not run:
important_nodes <- lapply(seq_len(k), function(i) {
ni <- node_importance_by_cluster(ci$state_df, clusters)
subset(ni, cluster == i & importance > 0.5)
})
plot_clustered_phylo(phy, ci$state_df, clusters, res$states, important_nodes,
cluster_colors = cols, state_colors = state_cols)

## End(Not run)
```

`plot_raw_recon` *Raster plot of a sample of joint reconstructions*

Description

Plots a matrix of sampled joint ancestral state reconstructions as a raster, with one row per reconstruction and one column per internal node. When cluster assignments are supplied, reconstructions are sorted by cluster and a colour bar identifies each cluster, making the "reconstruction axes" recovered by clustering directly visible.

Usage

```
plot_raw_recon(state_matrix, cluster_assignments = NULL, cluster_colors = NULL,
state_colors = NULL, state_labels = NULL, show_legend = TRUE,
xlab = "Node", ylab = "Reconstruction", main = NULL, ...)
```

Arguments

`state_matrix`	A matrix (or data.frame) of sampled reconstructions, rows being reconstructions and columns being internal nodes, with entries coded as integer states `1:k`. Typically `as.matrix(x$state_df)` from `compute_joint_ci`.
`cluster_assignments`	Optional integer vector of cluster memberships, one per row of `state_matrix`. A value of `0` marks an unclustered (noise) reconstruction; these are drawn last in grey. If `NULL`, all rows are drawn in their original order.
`cluster_colors`	Optional vector of colours for the clusters. Defaults to `hcl.colors(..., "Dark 3")`.
`state_colors`	Optional vector of colours for the character states. Defaults to `hcl.colors(k, "Zissou 1")`.
`state_labels`	Optional character vector of state names used in the legend. Defaults to `1:k`.
`show_legend`	Logical. Draw a state legend. Default `TRUE`.
`xlab, ylab, main`	Standard plot labels.
`...`	Further arguments passed to `image`.

Value

Invisibly returns a list with `order`, the row ordering used, and `cluster`, the cluster assignments in that order.

Author(s)

James D. Boyko

See Also

`compute_joint_ci`, `plot_clustered_phylo`

Examples

```
 ## Not run:
ci <- compute_joint_ci(res)
plot_raw_recon(as.matrix(ci$state_df))

## End(Not run)
```

plot_stacked_densities

Stacked log-likelihood densities by cluster

Description

Draws one density curve per cluster of sampled joint reconstructions, stacked vertically on a shared log-likelihood axis. Clusters whose densities overlap are statistically comparable alternative histories; a cluster shifted well to the left is a worse explanation of the data.

Usage

```
plot_stacked_densities(lnliks, clusters, cols, cluster_labels = NULL,
best_lnlik = NULL, xlab = "log-likelihood", ...)
```

Arguments

`lnliks`	Numeric vector of log-likelihoods, one per sampled reconstruction, as returned in `lnliks` by `compute_joint_ci`.
`clusters`	Vector of cluster assignments, one per reconstruction. Reconstructions assigned to cluster `0` (noise) are dropped.
`cols`	Vector of fill colours, one per cluster.
`cluster_labels`	Optional character vector of names for the clusters, drawn on the left axis. Defaults to the cluster numbers.
`best_lnlik`	Optional log-likelihood of the best joint reconstruction, drawn as a dashed vertical reference line.
`xlab`	Label for the log-likelihood axis.
`...`	Further arguments passed to `plot`, e.g. `main`.

Value

Called for its side effect. Returns `NULL` invisibly (and with a message) if only one cluster is present.

Author(s)

James D. Boyko

See Also

`compute_joint_ci`

Examples

```
 ## Not run:
plot_stacked_densities(ci$lnliks, clusters, cols = cluster_colors)

## End(Not run)
```

plot_transition_summary

Plot Summary Statistics of Stochastic Character Map Transitions

Description

Creates a two-panel plot to visualize the results from a list of stochastic character mapping simulations. The function generates a consolidated boxplot for transition counts and a staggered density plot (ridgeline plot) for transition ages.

Usage

```
plot_transition_summary(simmap_summaries, cols = NULL)
```

Arguments

`simmap_summaries`
: A list of data frames, where each data frame is the result of applying `summarize_single_simmap` to a "simmap" object.

`cols`
: An optional vector of character strings specifying colors for the plots. If `NULL`, a default color palette is used. Colors are recycled if there are more transition types than colors provided.

Details

This function produces a visualization of simulation results from stochastic character mapping. It is designed to be used after processing a list of "simmap" objects with `lapply(simmaps, summarize_single_simmap)`.

The graphical output consists of two main panels:

- **Left Panel:** A single plot containing vertical boxplots for the count of each transition type, summarized across all simulations. Semi-transparent points are overlaid with a small amount of jitter to show the distribution of individual data points.
- **Right Panel:** A ridgeline plot showing the distribution of transition ages.

Examples

```
## Not run:
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]

##run corhmm
MK <- corHMM(phy, data, 1)

##get simmap from corhmm solution
model <- MK$solution
simmaps <- makeSimmap(tree=phy, data=data, model=model, rate.cat=1, nSim=100, nCores=1)
```

```
# get the summary
simmap_summaries <- lapply(simmaps, summarize_single_simmap)
summary_df <- summarize_transition_stats(simmap_summaries)
print(summary_df)

plot_transition_summary(simmap_summaries)

## End(Not run)
```

---

plotDredgeTrace *Plot the Trace of a corHMMDredge Simulated Annealing Search*

Description

This function visualizes the search history of a corHMMDredge run. It creates a trace plot showing the information criterion score at each iteration, distinguishes between accepted and rejected models, and displays the type of move proposed at each step.

Usage

```
plotDredgeTrace(dredge_fits, break_size = 5,
               palette = c(drop = "#A23B72", merge = "#2E86AB", free = "#F18F01",
                 restart = "#7209B7", lump = "#C73E1D",
                 collapse = "#3B7A57", none = "grey60"),
                 legend = TRUE, legend.pos = "topright", ...)
```

Arguments

dredge_fits	An object of class corhmm.dredge returned by corHMMDredge with the argument return.all=TRUE.
break_size	An integer specifying the size of the empty gap in the plot between the traces of different rate categories.
palette	A named vector of colors for the different move types ("drop", "merge", "free", "lump", "collapse", "restart", "none"). The function will add default colors for any unpecified move types found in the data.
legend	Logical. If TRUE, a legend is added to the plot.
legend.pos	A character string specifying the position of the legend (e.g., "topright", "bottomleft").
...	Additional arguments to be passed to the main plot function, such as main, xlab, ylab, etc.

Details

The plot provides a comprehensive overview of the simulated annealing search process. The main plot area shows the information criterion score (e.g., AIC) on the y-axis against the iteration number on the x-axis. Each point represents a model that was evaluated.

- **Green circles** denote proposed models that were accepted.
- **Red 'x's** denote proposed models that were rejected.
- A **grey line** connects the sequence of accepted models to show the path of the search.
- A **colored bar** at the bottom of the plot indicates the type of move proposed at each iteration ("drop", "merge", "free", or "restart").
- If the dredge search was performed over multiple rate categories (i.e., `max.rate.cat > 1`), the traces for each rate category run are separated by a **dashed vertical line**.

This visualization is useful for diagnosing the performance of the search, such as assessing whether the temperature and cooling schedule were appropriate, or if the search became trapped in a local optimum.

Author(s)

James D. Boyko

See Also

`corHMMDredge`

---

plotMKmodel *Plot a Markov model*

Description

Plots a diagram of a Markov model from the output of corHMM or a custom index matrix

Usage

```
plotMKmodel(corhmm.obj, rate.cat = NULL, display = "column", color = c("blue", "red"),
arrow.scale = 1, text.scale = 1, vertex.scale = 1)
```

Arguments

`corhmm.obj`	an object of class corHMM or matrix.
`rate.cat`	if using a custom matrix then the number of rate categories must be indicated.
`display`	the structure of the plot. one of "column", "square", or "row".
`color`	Either, 1. a vector of 2 colors to create a gradient from low transition rates (first element) to high transition rates (second element), or 2. "col.blind" which will use the color pallete "plasma" from viridis.
`arrow.scale`	determines the size of the arrows for the Markov diagram.
`text.scale`	determines the size of the text for the plotted matrix.
`vertex.scale`	determines the size of the text for the Markov diagram.

Details

Plots Markov models in a ball and stick type diagram next to its corresponding matrix. If plotting a hidden rates model it will produce a compound plot describing how the different rate classes are related to one another. If the input is a `corHMM` result then arrows are colored by relative rate. If the input is a custom matrix arrows are colored by the paramater index.

Value

Returns a ball and stick diagram of the input model.

Author(s)

James D. Boyko

References

Boyko, J. D., and J. M. Beaulieu. 2021. Generalized hidden Markov models for phylogenetic comparative datasets. Methods in Ecology and Evolution 12:468-478.

Examples

```
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
# create a legend and rate mat from a multi-character dataset.
LegendAndRateMat <- getStateMat4Dat(data)
rate.mat <- LegendAndRateMat$rate.mat
legend <- LegendAndRateMat$legend

# To create a hidden markov model first define your rate classes (state-dependent processes)
# R1 will be a manually created SYM model
R1 <- equateStateMatPars(rate.mat, c(1:6))
# R2 will only allow transitions between 1 and 2
R2 <- dropStateMatPars(rate.mat, c(3,4))
# R1 and R2 will transtion at equal rates (i.e. the parameter process will be ER)
P <- getRateCatMat(2)
P <- equateStateMatPars(P, c(1,2))
# combine our state-dependnet processes and parameter process
HMM <- getFullMat(list(R1, R2), P)
# plot the input
plotMKmodel(HMM, rate.cat = 2)

# This can now be used in a corHMM model
CustomModel <- corHMM(phy = phy, data = data, rate.cat = 2, rate.mat = HMM, node.states = "none")
# plot the output
plotMKmodel(CustomModel)
```

plotRECON *Plot ancestral state reconstructions*

Description

Plots maximum likelihood ancestral state estimates on tree

Usage

```
plotRECON(phy, likelihoods, piecolors=NULL, cex=0.5, pie.cex=0.25, file=NULL,
height=11, width=8.5, show.tip.label=TRUE, title=NULL, ...)
```

Arguments

phy	a phylogenetic tree, in ape "phylo" format.
likelihoods	likelihoods for ancestral states (see Details).
piecolors	a vector of colors for states.
cex	specifies the size of the font for labels (if used).
pie.cex	specifies the size of the symbols to plot on tree.
file	filename to which a pdf is saved.
height	height of plot.
width	width of plot.
show.tip.label	a logical indicating whether to draw tip labels to tree. The default is TRUE.
title	an optional title for the plot.
...	Additional arguments to be passed to the plot device

Details

Plots ancestral state estimates on provided tree. The likelihoods can be the states of an object of class rayDISC or class corDISC, or the lik.anc of an object of class ace (from the ape package).

Value

A plot indicating the maximum likelihood ancestral states at each internal node.

Author(s)

Jeffrey C. Oliver

See Also

corDISC, rayDISC

Examples

```
data(rayDISC.example)
## Perform ancestral state estimation, using a single rate of evolution and marginal
## reconstruction of ancestral states
recon <- rayDISC(rayDISC.example$tree,rayDISC.example$trait,model="ER",
node.states="marginal")
## Plot reconstructions on tree
plotRECON(rayDISC.example$tree,recon$states,title="rayDISC Example")
```

---

print.dentist — *Print dentist print summary of output from dent_walk*

Description

Print dentist print summary of output from dent_walk

Usage

```
## S3 method for class 'dentist'
print(x, ...)
```

Arguments

x	An object of class dentist
...	Other arguments (not used)

---

rayDISC — *Evolution of categorical traits*

Description

Fits a model of evolution for categorical traits, allowing for multi-state characters, polymorphisms, missing data, and incompletely resolved trees

Usage

```
rayDISC(phy,data, ntraits=1, charnum=1, rate.mat=NULL, model=c("ER","SYM","ARD"),
node.states=c("joint", "marginal", "scaled", "none"), state.recon=c("subsequently"),
lewis.asc.bias=FALSE, p=NULL, root.p="yang", ip=NULL, lb=1e-9, ub=100, verbose=TRUE,
diagn=FALSE)
```

Arguments

phy	a phylogenetic tree, in ape "phylo" format.
data	a data matrix containing species information (see Details).
ntraits	specifies the number of traits to included in the analysis.
charnum	specified the character to analyze.
rate.mat	a user-supplied rate matrix index of parameters to be optimized.
model	specifies the underlying model.
node.states	method used to calculate ancestral states at internal nodes.
state.recon	whether to reconstruct states jointly with the rates or subsequent to the rates being optimized.
lewis.asc.bias	a logical indicating whether the ascertainment bias correction of Lewis et al. 2001 should be used. The default is FALSE.
p	a vector of transition rates. Allows the user to calculate the likelihood given a specified set of parameter values to specified as fixed and calculate the likelihood.
root.p	a vector used to fix the probabilities at the root, but "yang" and "maddfitz" can also be supplied to use the method of Yang (2006) and FitzJohn et al (2009), respectively (see details).
ip	initial values used for the likelihood search. Can be a single value or a vector of unique values for each parameter. The default is ip=1.
lb	lower bound for the likelihood search. The default is lb=0.
ub	upper bound for the likelihood search. The default is ub=100.
verbose	a logical indicating whether progress should be printed to the screen.
diagn	logical indicating whether diagnostic tests should be performed. The default is FALSE.

Details

__THIS FUNCTION IS NO LONGER NECESSARY AS IT IS NOW ENTIRELY SUBSUMED WITHIN__ corHMM (see _Generalized corHMM_ vignette). But we still provide it for those that are more comfortable using it than exploring the new corHMM function. As before, rayDISC takes a tree and a trait file and estimates transition rates and ancestral states for binary or multistate characters. The first column of the trait file must contain the species labels to match to the tree, with the second, third, fourth, and so on, corresponding to the traits of interest. Use the charnum variable to select the trait for analysis. Also, the input phylogeny need not be bifurcating as the algorithm is implemented to handle multifucations. Polytomies are allowed by generalizing Felsenstein's (1981) pruning algorithm to be the product of the probability of observing the tip states of n descendant nodes, rather than two, as in the completely bifurcating case.

The user can fix the root state probabilities by supplying a vector to the root.p. If the user supplies the flag root.p="yang", then the estimated transition rates are used to set the weights at the root (see pg. 124 Yang 2006), whereas specifying root.p="maddfitz" employs the same procedure described by Maddison et al. (2007) and FitzJohn et al. (2009). Note that the default root.p=NULL assumes equal weighting among all possible states.

Ambiguities (polymorphic taxa or taxa missing data) are assigned likelihoods following Felsenstein (2004, p. 255). Taxa with missing data are coded "?" with all states observed at a tip. Polymorphic taxa are coded with states separated by an "&". For example, if a trait has four states and taxonA is observed to be in state 1 and 3, the character would be coded as "1&3". corHMM then uses this information to assign a likelihood of 1.0 to both states. Missing data are treated as ambiguous for all states, thus all states for taxa missing data are assigned a likelihood of 1.0. For example, for a four-state character (i.e. DNA), a taxon missing data will have likelihoods of all four states equal to 1.0 [e.g. L(A)=1.0, L(C)=1.0, L(G)=1.0, L(T)=1.0].

In all ancestral state reconstruction implementations, the rates are first estimated, and subsequently, the MLE estimates of the rates are used to determine either the state probabilities (i.e., marginal or "scaled") or maximum likelihood states at nodes. This is the default – i.e., the state.recon="subsequently" argument. However, for this function only, we also allow for both rates and states to be estimated jointly. This can be done with state.recon="estimate". We also allow for a hypothesis about states at all or even some nodes to help fixed, with the rates (and in some cases some of the states) being estimated. This is state.recon="given". For more information please see Vignette "Getting Likelihoods From Reconstructions".

Value

`rayDISC` returns an object of class `rayDISC`. This is a list with elements:

`$loglik`	the maximum negative log-likelihood.
`$AIC`	Akaike information criterion.
`$AICc`	Akaike information criterion corrected for sample size.
`$ntraits`	The number of traits specified.
`$solution`	a matrix containing the maximum likelihood estimates of the transition rates.
`$solution.se`	a matrix containing the approximate standard errors of the transition rates. The standard error is calculated as the square root of the diagonal of the inverse of the Hessian matrix.
`$index.mat`	The indices of the parameters being estimated are returned. The numbers correspond to the row in the `eigvect` and can useful for identifying the parameters that are causing the objective function to be at a saddlepoint.
`$lewis.asc.bias`	The setting describing whether or not the Lewis ascertainment bias correction was used.
`$opts`	Internal settings of the likelihood search.
`$data`	User-supplied dataset.
`$phy`	User-supplied tree.
`$states`	The likeliest states at each internal node.
`$tip.states`	NULL
`$iterations`	The number of iterations used by the optimization routine.
`$eigval`	The eigenvalues from the decomposition of the Hessian of the likelihood function. If any `eigval<0` then one or more parameters were not optimized during the likelihood search.

`$eigvect`	The eigenvectors from the decomposition of the Hessian of the likelihood function is returned.
`$bound.hit`	A logical for diagnosing if rate parameters were constrained by `lb` or `ub` values during optimization.
`$message.tree`	A list of taxa which were listed in the data matrix, but were not present in the passed `phylo` object. These taxa will be excluded from the analysis. `message.tree` is null if all taxa in data are included in tree.
`$message.data`	A list of taxa which were present in the passed `phylo` object, but lacked data in the passed data matrix. These taxa will be coded as missing data (all states equally likely). `message.data` is null if all taxa in tree have entries in data matrix.

Author(s)

Jeffrey C. Oliver and Jeremy M. Beaulieu

References

Felsenstein, J. 1981. A likelihood approach to character weighting and what it tells us about parsimony and compatibility. Biological Journal of the Linnean Society 16: 183-196.

Felsenstein J. 2004. Inferring phylogenies. Sunderland MA: Sinauer Associates.

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

Lewis, P.O. 2001. A likelihood approach to estimating phylogeny from discrete morphological character data. Systematic Biology 50:913-925.

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

See Also

`plotRECON`

Examples

```
### Example 1
data(rayDISC.example)
## Perform ancestral state estimation, using an asymmetric model of evolution and marginal
## reconstruction of ancestral states
recon <- rayDISC(rayDISC.example$tree,rayDISC.example$trait,model="ARD",
node.states="marginal")

## Plot reconstructions on tree
plotRECON(rayDISC.example$tree,recon$states)

### Example 2
## Perform ancestral state estimation on second character, using a single-rate model of
## evolution, marginal reconstruction of ancestral states, and setting the lower bound for
## parameter estimates to 0.01
```

```
recon <- rayDISC(rayDISC.example$tree,rayDISC.example$trait,charnum=2,model="ER",
node.states="marginal",lb=0.01)

### Example 3
## Perform ancestral state estimation on third character, using a single-rate model of
## evolution and joint reconstruction of ancestral states
recon <- rayDISC(rayDISC.example$tree,rayDISC.example$trait,charnum=3,
model="ER",node.states="joint")
```

`simMarkov` *Simulate a character on the tree*

Description

Simulates a character using an instanteous rate matrix, storing the states at nodes and at the tips of the tree.

Usage

```
simMarkov(phy, Q, root.freqs)
```

Arguments

`phy`	A phylogeny of class phylo.
`Q`	an instantaneous rate matrix.
`root.freqs`	A vector of root frequencies.

Details

This function will generate a character at tips and at interior nodes given a tree and Q matrix.

Value

A two element list containing ‘TipStates‘ and ‘NodeStates‘

Author(s)

James D. Boyko

summarize_transition_stats

Summarize Transition Information from Stochastic Character Maps

Description

These functions process and summarize transition events (e.g., character state changes) from one or more stochastic character maps, such as those generated by `makeSimmap`. `summarize_single_simmap` extracts a detailed data frame of all transitions from a single stochastic map object. `summarize_transition_stats` takes a list of these data frames (from multiple stochastic maps) and computes summary statistics, such as the average number and timing of each transition type.

Usage

```
summarize_single_simmap(simmap)
summarize_transition_stats(simmap_summaries)
```

Arguments

`simmap` An object of class "simmap", representing a single phylogenetic tree with a character history mapped onto it. Typically an element from the list produced by `makeSimmap`.

`simmap_summaries` A list of data frames, where each data frame is the result of applying `summarize_single_simmap` to a "simmap" object.

Details

The intended workflow involves two main steps. First, one uses `lapply` to apply `summarize_single_simmap` to each element in a list of stochastic maps (e.g., the output of `makeSimmap(..., nsim=100)`). This creates a list of data frames, with each data frame containing the transition history for one simulation. Second, this list of data frames is passed to `summarize_transition_stats` to be aggregated into a final summary table that provides means and standard deviations for transition counts and timings across all simulations.

Value

`summarize_single_simmap` returns a data frame with three columns for a single simulation:

`branch_id` The integer ID of the branch on which the transition occurred.

`transition` A character string describing the transition, e.g., "state1,state2".

`age` The time at which the transition occurred, measured from the root of the tree.

`summarize_transition_stats` returns a single data frame summarizing all simulations, with the following columns:

`transition` The unique transition type.

avg_count	The average number of times this transition occurred per simulation.
sd_count	The standard deviation of the count for this transition across simulations.
avg_age	The average time (age) at which this transition occurred, across all occurrences in all simulations.
sd_age	The standard deviation of the time (age) for this transition.

Examples

```
## Not run:
data(primates)
phy <- primates[[1]]
phy <- multi2di(phy)
data <- primates[[2]]
## run corhmm
MK <- corHMM(phy, data, 1)
## get simmap from corhmm solution
model <- MK$solution
simmaps <- makeSimmap(tree=phy, data=data, model=model, rate.cat=1, nSim=100, nCores=1)
## get the summary
simmap_summaries <- lapply(simmaps, summarize_single_simmap)
summary_df <- summarize_transition_stats(simmap_summaries)
print(summary_df)

## End(Not run)
```

---

summary.dentist *Summarize dentist Display summary of output from dent_walk*

Description

Summarize dentist Display summary of output from dent_walk

Usage

```
## S3 method for class 'dentist'
summary(object, ...)
```

Arguments

object	An object of class dentist
...	Other arguments (not used)

`tipHomology` *The Probability of a Tip State Being Homologous*

Description

This function takes a fitted model (in the form of a corHMM object) and returns the probability that different tips are homologous with respect to a specified node. Two types of "homology" are included: the first is the probability that there was no change throughout the lineage's entire history, and the second is the probability that the starting and ending states were the same.

Usage

```
tipHomology(corhmm_obj, type = "strict", node = NULL, return.likelihoods = FALSE)
```

Arguments

`corhmm_obj` A corHMM fit returned from `corHMM`.

`type` Either "strict" or "wagner". "strict" specifies the probability of no change throughout a lineage's history, while "wagner" returns the probability of starting and ending in the same state.

`node` The node number from which to calculate the probability of homology. If NULL, the root is used. Only tips descending from the node are returned (a tip state cannot be homologous with a node it never descended from).

`return.likelihoods` Whether to return the exact probability of the scenario or the probability relative to the likelihood of the dataset and model.

Details

The probability of homology is defined as either the probability that a tip inherited its state without ever changing throughout its history ("type = strict") or that the specified ancestor and the tip had the same state ("type = wagner").

In the first case, the probability is calculated by fixing all branches between the tip and the specified node to a particular state. For example, if the tip state is 0, all branches between the tip and specified node are treated as being in state 0. Other branches not between the focal tip and the specified ancestral node are treated as unknown.

In the "wagner" homology case, the probability is calculated as the likelihood of the tip and the ancestor sharing the same state. For example, if the tip state is 0, this corresponds to assuming that all nodes along the path between the tip and the ancestral node are also in state 0.

To build intuition, consider the probability of homology as being similar to ancestral state reconstruction. It integrates over all possibilities for branches and nodes not explicitly constrained. Another perspective is to interpret these probabilities as inversely proportional to the number of trait changes. This is particularly intuitive in the "strict" case, where the probability directly reflects the likelihood of no changes occurring along the lineage.

One could use a likelihood ratio test (LRT) to evaluate evidence for at least one change (using `chisq` with one degree of freedom). However, since the null hypothesis assumes no change, the LRT can only fail to reject strict homology; it cannot provide evidence for significant homology.

Value

Returns a vector of homology probabilities.

Author(s)

James D. Boyko

Index

∗ **datasets**
examples, 29
∗ **models**
compute_joint_ci, 6
ComputeCI, 8
corDISC, 10
corHMM, 12
corHMMDredgeBase, 23
getFullMat, 33
getStateMat4Dat, 36
node_importance_by_cluster, 42
phylo_aware_dist_fast, 43
phylo_aware_dist_local, 44
rayDISC, 57
simMarkov, 61
∗ **plot**
plot_batch_profile_lik, 46
plot_clustered_phylo, 48
plot_raw_recon, 49
plot_stacked_densities, 51
plotMKmodel, 54
plotRECON, 56
∗ **reconstructions**
ancRECON, 2
ancRECON_slice, 5

ancRECON, 2
ancRECON_slice, 5

compute_joint_ci, 6, *43–45*, *48–51*
ComputeCI, 8
ConvertPhangornReconstructions, 9
corDISC, 10, *56*
corHMM, 12, *18*, *21*, *23–25*, *32*, *36*
corHMMDredge, 17, *23–25*, *53, 54*
corHMMDredgeBase, 23

dent_likelihood, 26
dent_propose, 26
dent_walk, 27
dev.raydisc *(rayDISC)*, 57
dropStateMatPars *(getFullMat)*, 33

equateStateMatPars *(getFullMat)*, 33
examples, 29

fitCorrelationTest, 29

get_batch_profile_lik, 31
getCVTable, 33
getFullMat, 33
getModelTable, *21*, 35
getRateCatMat *(getFullMat)*, 33
getStateMat4Dat, 36

kFoldCrossValidation, *21*, 38

makeSimmap, 39

node_importance_by_cluster, 42, *45*, *48, 49*

phylo_aware_dist_fast, 43
phylo_aware_dist_local, *43*, 44, *44*
plot.dentist, 46
plot_batch_profile_lik, 46
plot_clustered_phylo, *43*, 48, *50*
plot_raw_recon, 49, *49*
plot_stacked_densities, 51
plot_transition_summary, 52
plotDredgeTrace, *21*, 53
plotMKmodel, 54
plotRECON, 56, *60*
primates *(examples)*, 29
print.dentist, 57

rayDISC, *56*, 57
rayDISC.example *(examples)*, 29

simMarkov, 61

summarize_single_simmap
(summarize_transition_stats), 62
summarize_transition_stats, 62
summary.dentist, 63

tipHomology, 64