

Package: evobiR (via r-universe)

October 15, 2026

Type Package

Title Evolutionary Biology in R

Version 2.2.1

Date 2026-08-21

Author Michelle M. Jonika, Maximos Chin, Nathan Anderson, Meghann McConnell, Richard H. Adams, Jeffery P. Demuth, and Heath Blackmon

Maintainer Heath Blackmon <coleoguy@gmail.com>

URL <https://github.com/coleoguy/evobir>

BugReports <https://github.com/coleoguy/evobir/issues>

Description Comparative analysis of continuous traits influencing discrete states, and utility tools to facilitate comparative analyses. Implementations of ABBA/BABA type statistics to test for introgression in genomic data.

License GPL (>=2)

Imports seqinr, ape, phytools, castor, methods, viridis, RColorBrewer, expm, igraph

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.3

Encoding UTF-8

Config/pak/sysreqs libglib-dev make libxml2-dev zlib1g-dev

Repository <https://phylostatic.r-universe.dev>

Date/Publication 2026-09-01 18:26:22 UTC

RemoteUrl <https://github.com/coleoguy/evobir>

RemoteRef HEAD

RemoteSha ec8e0587a97c4b5f916118e88be247494d2c3a07

Contents

evobiR-package	2
1.fasta	3
AncCond	4
CalcD	6
CalcPopD	8
countTrees	9
fix.simmap	11
FuzzyMatch	12
getNe	13
GetTipRates	14
horn.beetle.csv	16
hym.tree	16
make.simmap2	17
mite.trait	18
Mode	19
plot.phylocaled	19
pSAF	20
ReOrderAlignment	23
ResSel	24
SampleTrees	26
scaleTreeRates	27
SlidingWindow	28
SuperMatrix	30
trees.mite	31
trees.nex	31
WinCalcD	32
Index	34

evobiR-package

evobiR: Evolutionary Biology in R

Description

evobiR is a collection of tools for use in evolutionary biology. Functions include comparative methods for analyzing the relationship between continuous and discrete traits on phylogenies, implementations of Patterson's D-statistic for testing introgression, tools for manipulating sequence alignments and phylogenetic trees, and utility functions for sliding window analyses, residual selection, and population genetics calculations.

Details

Package: evobiR
Type: Package
Version: 2.2
Date: 2025-02-20
License: GPL (>=2)

More information on **evobiR** is available at <https://github.com/coleoguy/evobir>

Author(s)

Michelle M. Jonika, Maximos Chin, Nathan Anderson, Richard H. Adams, Jeffery P. Demuth, and Heath Blackmon

Maintainer: Heath Blackmon <coleoguy@gmail.com>

1.fasta

Simulated SNP alignment data

Description

Simulated SNP alignment files in FASTA format for use with the D-statistic functions.

1.fasta contains 4 sequences representing single individuals in the order P1, P2, P3, and Outgroup, for use with [CalcD](#) and [WinCalcD](#).

3.fasta contains population-level data with multiple sequences per population (sequences from the same population share identical names), for use with [CalcPopD](#).

Author(s)

Heath Blackmon

References

<http://coleoguy.github.io/>

AncCond

*Calculate the mean of a continuous character at transitions in a binary character***Description**

Tests whether transitions in a binary discrete trait are associated with extreme ancestral values of a continuous trait. Null distribution generated by holding the continuous trait fixed and simulating discrete traits under the fitted Mk model. The continuous and discrete traits may each be observed on any subset of the tips, including disjoint subsets.

Usage

```
AncCond(tree,
        x,
        y,
        n.maps = 100,
        n.sims = 1000,
        prune = FALSE,
        Q = NULL,
        hypothesis = "none")
```

Arguments

tree	tree of class phylo
x	named numeric vector: continuous trait at tips. Tips may be absent from x or coded NA; at least 3 tips must carry data.
y	named integer vector: binary discrete trait (0/1) at tips. Tips may be absent from y or coded NA; both states must be observed in at least one tip. x and y need not be observed on the same tips.
n.maps	number of stochastic maps per test-statistic computation
n.sims	number of null simulations
prune	if TRUE, estimate the continuous trait using only lineages observed in state 0 (tips with unknown y are excluded as well)
Q	optional 2x2 rate matrix (rows/cols labelled "0","1"). If supplied, the Mk model is NOT fitted; the provided rates are used directly for stochastic mapping and null simulation. Useful when the user has a preferred model or wants to avoid automatic model selection.
hypothesis	direction of the alternative hypothesis. One of: "none": two-sided test (default): $p = \min(1, 2 * \min(p.\text{upper}, p.\text{lower}))$ "higher": one-sided: transitions occur at HIGH continuous values ($p = p.\text{upper}$) "lower": one-sided: transitions occur at LOW continuous values ($p = p.\text{lower}$)

Details

This function uses ancestral state estimates to determine if the transitions in the binary trait are associated with extreme values of the continuous trait. This test can incorporate the possibility that the derived state of a binary character may lead to correlated selection in the continuous trait. If this is desired then `prune = TRUE` restricts the ancestral state estimation of the continuous trait to lineages observed in state 0.

Missing data are handled as follows. Tips without `y` are marginalised out of the Mk likelihood and of the stochastic maps (their state is sampled from the posterior given the rest of the data), and the null simulations are masked to the same missingness pattern so that observed and null test statistics are computed from equivalent information. Tips and internal nodes that lack a direct continuous-trait estimate receive their Brownian motion conditional expectation given the tips that carry `x`: nodes lying on an edge of the tree pruned to the `x`-bearing tips are linearly interpolated between the estimates at the ends of that edge (weighted by branch length), and nodes in subtrees with no `x` data take the value at the point where that subtree attaches. Transitions inferred in lineages without `x` therefore contribute an imputed value, and power falls as the overlap between the `x`- and `y`-bearing tips shrinks because the signal then comes only through phylogenetic signal in `x`.

Value

Returns a list of class `anccond` with elements:

<code>T.obs.01</code>	observed mean continuous value at 0 to 1 transitions
<code>T.obs.10</code>	observed mean continuous value at 1 to 0 transitions
<code>p.01</code>	p-value for 0 to 1 transitions (adjusted per hypothesis)
<code>p.10</code>	p-value for 1 to 0 transitions (adjusted per hypothesis)
<code>p.combined</code>	Bonferroni-corrected p-value across both transition directions: $\min(1, 2 * \min(p.01, p.10))$. Use this when interpreting both <code>p.01</code> and <code>p.10</code> as independent tests.
<code>p.upper.01</code>	one-sided $P(T_{\text{null}} \geq T_{\text{obs}})$ for 0 to 1
<code>p.lower.01</code>	one-sided $P(T_{\text{null}} \leq T_{\text{obs}})$ for 0 to 1
<code>p.upper.10</code>	one-sided $P(T_{\text{null}} \geq T_{\text{obs}})$ for 1 to 0
<code>p.lower.10</code>	one-sided $P(T_{\text{null}} \leq T_{\text{obs}})$ for 1 to 0
<code>T.null.01</code>	null distribution for 0 to 1 (length <code>n.sims</code>)
<code>T.null.10</code>	null distribution for 1 to 0 (length <code>n.sims</code>)
<code>Q</code>	fitted rate matrix
<code>model</code>	"ARD", "unidirectional", or "user-supplied"
<code>hypothesis</code>	hypothesis direction used
<code>n.obs</code>	named vector: number of tips in the tree, with <code>x</code> observed, and with <code>y</code> observed

Author(s)

Meghann McConnell, Nathan Anderson, Richard H. Adams, and Heath Blackmon

References

<http://coleoguy.github.io/>

Examples

```
## Not run:
# Simulate a toy dataset
set.seed(1)
tree <- pbtree(n = 100, scale = 1)
x <- fastBM(tree) # continuous trait
y <- rbinom(100, 1, 0.3); names(y) <- tree$tip.label # binary trait

# Run the test (small settings for a quick demo)
res <- AncCond(tree, x, y, n.maps = 50, n.sims = 500)

# Continuous and discrete traits measured on different species
res2 <- AncCond(tree, x[1:60], y[41:100], n.maps = 50, n.sims = 500)

## End(Not run)
```

CalcD

Patterson's D-Statistic (ABBA-BABA Test)

Description

Calculates Patterson's D-statistic to test for introgression (gene flow) between species using genomic sequence data. The test looks at two patterns of shared alleles – called "ABBA" and "BABA" – that should be equally common if there is no introgression. A significant departure from $D = 0$ suggests gene flow between non-sister taxa.

Usage

```
CalcD(
  alignment = "alignment.fasta",
  sig.test = "N",
  ambig = "D",
  block.size = 1000,
  replicate = 1000,
  align.format = "fasta"
)
```

Arguments

alignment	Character string. Path to the alignment file (default "alignment.fasta").
sig.test	Character. Type of significance test: • "N" (default) – no significance test, just calculate D. • "B" – bootstrap: resample sites with replacement. • "J" – jackknife: drop blocks of sites. Good when nearby SNPs may be in linkage disequilibrium.
ambig	Character. How to handle ambiguous bases (e.g., R, Y, S):

	• "D" (default) – drop sites with any ambiguity. • "R" – randomly resolve each ambiguous base. • "I" – ignore ambiguities (keep them as-is).
block.size	Integer. Number of contiguous sites to drop in each jackknife replicate (default 1000). Only used when sig.test = "J".
replicate	Integer. Number of bootstrap or jackknife replicates (default 1000).
align.format	Character. Format of the alignment file (default "fasta"). Any format accepted by read.alignment .

Details

The input alignment must contain exactly 4 sequences in this order:

1. **P1** – first ingroup population
2. **P2** – second ingroup population (suspected introgression recipient)
3. **P3** – third population (suspected introgression donor)
4. **O** – outgroup

D is calculated as $(ABBA - BABA) / (ABBA + BABA)$.

- D = 0 means no evidence of introgression.
- D > 0 suggests gene flow between P2 and P3.
- D < 0 suggests gene flow between P1 and P3.

Only biallelic sites are used. Significance is assessed by computing a Z-score ($|D| / SD$) from the resampled distribution, then converting to a two-tailed p-value.

Value

The D-statistic value (numeric). Also prints summary statistics to the console including site counts and (if requested) p-values.

References

Durand, E.Y., Patterson, N., Reich, D., & Slatkin, M. (2011). Testing for ancient admixture between closely related populations. *Molecular Biology and Evolution*, 28(8), 2239-2252.

Examples

```
## Not run:
# Basic D-statistic with no significance test
CalcD("my_alignment.fasta")

# With bootstrap significance test
CalcD("my_alignment.fasta", sig.test = "B", replicate = 500)

# With jackknife (good for whole-genome data with LD)
CalcD("my_alignment.fasta", sig.test = "J", block.size = 5000)

## End(Not run)
```

CalcPopD

*Population-Level Patterson's D-Statistic***Description**

Like [CalcD](#), but designed for population-level data where you have multiple individuals per population. Instead of counting discrete ABBA/BABA patterns, it uses allele frequencies to compute a frequency-based D-statistic (Equation 2 from Durand et al. 2011).

Usage

```
CalcPopD(
  alignment = "alignment.fasta",
  sig.test = "N",
  ambig = "D",
  block.size = 1000,
  replicate = 1000,
  align.format = "fasta"
)
```

Arguments

<code>alignment</code>	Character string. Path to the alignment file (default "alignment.fasta").
<code>sig.test</code>	Character. Type of significance test: • "N" (default) – no significance test, just calculate D. • "B" – bootstrap: resample sites with replacement. • "J" – jackknife: drop blocks of sites. Good when nearby SNPs may be in linkage disequilibrium.
<code>ambig</code>	Character. How to handle ambiguous bases (e.g., R, Y, S): • "D" (default) – drop sites with any ambiguity. • "R" – randomly resolve each ambiguous base. • "I" – ignore ambiguities (keep them as-is).
<code>block.size</code>	Integer. Number of contiguous sites to drop in each jackknife replicate (default 1000). Only used when <code>sig.test = "J"</code> .
<code>replicate</code>	Integer. Number of bootstrap or jackknife replicates (default 1000).
<code>align.format</code>	Character. Format of the alignment file (default "fasta"). Any format accepted by read.alignment .

Details

When should you use this instead of CalcD? Use CalcPopD when you have sequenced multiple individuals from each population (e.g., 10 humans, 10 chimps, 10 gorillas, 10 orangutans). The frequency-based approach extracts more information from your data because it considers how common each allele is within each population, rather than just looking at a single representative sequence.

The alignment must contain sequences grouped by population name. Sequences from the same population should share the same name in the FASTA header. Populations must appear in this order: P1, P2, P3, Outgroup.

For each biallelic site, the function calculates the frequency of the derived allele (B) in each population:

$$D = \frac{\sum [(1-\hat{p}_1)\hat{p}_2\hat{p}_3(1-\hat{p}_4) - \hat{p}_1(1-\hat{p}_2)\hat{p}_3(1-\hat{p}_4)]}{\sum [(1-\hat{p}_1)\hat{p}_2\hat{p}_3(1-\hat{p}_4) + \hat{p}_1(1-\hat{p}_2)\hat{p}_3(1-\hat{p}_4)]}$$

This frequency-based approach is more powerful than the single-sequence version when population samples are available because it uses all the information about allele frequencies rather than just presence/absence.

Value

A list with:

d.stat The D-statistic value.

pval P-value (only if sig.test != "N").

align.length Total number of sites in the alignment.

useful.sites Number of biallelic sites with ABBA/BABA signal.

seg.sites Number of useful sites still segregating within at least one population.

References

Durand, E.Y., Patterson, N., Reich, D., & Slatkin, M. (2011). Testing for ancient admixture between closely related populations. *Molecular Biology and Evolution*, 28(8), 2239-2252.

Examples

```
## Not run:
# Population-level D-statistic with bootstrap
result <- CalcPopD("population_alignment.fasta", sig.test = "B")
result$d.stat
result$pval

## End(Not run)
```

Description

Classifies each tree in a collection against a set of reference topologies and counts how many trees match each reference.

Usage

```
countTrees(collection = NULL, ref = NULL, classes = TRUE, verbose = TRUE)
```

Arguments

collection	Character. Path to a Newick format file containing the collection of trees to classify (e.g., posterior trees from an MCMC analysis).
ref	Character. Path to a Newick format file containing the reference topologies to count. Each tree in this file represents one possible branching pattern you want to test.
classes	Logical. If TRUE (default), returns both the counts per topology AND the classification of each individual tree. If FALSE, returns only the counts.
verbose	Logical. If TRUE (default), prints a warning listing any trees that don't match any reference topology.

Details

What is a "topology"? A topology is the branching pattern of a tree – which species are most closely related to which – ignoring branch lengths. For example, with three species (A, B, C), there are three possible topologies: ((A,B),C), ((A,C),B), and ((B,C),A). This function counts how many trees in your collection have each branching pattern.

When would you use this? After a Bayesian phylogenetic analysis (e.g., MrBayes, BEAST), you have thousands of sampled trees. This function tells you what fraction of those trees support each possible branching arrangement, helping you assess which topology has the most support in the posterior distribution.

Value

If classes = TRUE: a list with two elements:

[[1]] A numeric vector of counts – how many trees matched each reference topology.

[[2]] A numeric vector giving the topology class (reference number) for each tree in the collection. NA means the tree didn't match any of the reference topologies you provided.

If classes = FALSE: just the count vector.

Examples

```
## Not run:
# You have 1000 posterior trees and 3 candidate topologies.
# How many trees support each arrangement?
result <- countTrees("posterior_trees.nwk", "candidate_topologies.nwk")
result[[1]] # e.g., c(712, 201, 87) = topology 1 has most support
result[[2]] # which topology each individual tree matched

## End(Not run)
```

fix.simmap

*Fix Failed Branches in Stochastic Mapping***Description**

Repairs branches that failed during stochastic character mapping (from `make.simmap2` or `phytools::make.simmap`).

Usage

```
fix.simmap(hists, tips, transition.matrix)
```

Arguments

<code>hists</code>	A <code>simmap</code> or <code>multiSimmap</code> object (the output of <code>make.simmap2</code> or <code>make.simmap</code>) that may contain failed branches.
<code>tips</code>	A two-column data frame where: Column 1 Species names matching the tree tip labels. Column 2 The known discrete state for each species.
<code>transition.matrix</code>	A symmetric matrix describing which state transitions are possible. Non-zero entries indicate allowed transitions. This is used to find valid paths between states using shortest-path algorithms.

Details

What is a "failed" branch? Stochastic character mapping works by simulating trait evolution along each branch of a tree. For some branches, the simulation can't find a valid history that starts and ends at the correct states – like trying to find a path through a maze with too many dead ends. After many attempts (controlled by `rejmax` in `make.simmap2`), the branch is marked as "fail" instead of running forever.

This function repairs those failed branches by using graph theory (shortest path algorithm) to find the simplest valid sequence of state transitions between the start and end states, then divides the branch time equally among the required intermediate states.

For each failed branch, the function:

1. Identifies the start state (from the parent node) and end state (from the data or child node).
2. Finds the shortest valid path between these states using the transition matrix as a graph.
3. Divides the branch length equally among all states in the path.
4. Updates both the `maps` list and the `mapped.edge` matrix.

Value

A fixed `simmap` or `multiSimmap` object with the failed branches repaired. The `maps` and `mapped.edge` components are updated to reflect the inferred state transitions.

Examples

```
## Not run:
# Run stochastic mapping with a rejection limit
mapped_tree <- make.simap2(tree, states, model = "ARD",
                           nsim = 10, rejmax = 100000)
# Fix any branches that failed
fixed_tree <- fix.simap(mapped_tree, tip_data, trans_matrix)

## End(Not run)
```

FuzzyMatch

Fuzzy Name Matching Between Tree and Data

Description

Finds taxon names in your data that are close (but not exact) matches to the tip labels on your phylogenetic tree.

Usage

```
FuzzyMatch(tree, data, max.dist)
```

Arguments

tree	A phylogenetic tree of class "phylo" or "multiPhylo". If a multiPhylo object is provided, only the first tree is used (with a warning).
data	A character vector of taxon names from your dataset (e.g., the species column of your data frame).
max.dist	Integer. Maximum number of character differences to still count as a "close match." For example, max.dist = 2 will flag name pairs that differ by 1 or 2 characters.

Details

Why do you need this? In comparative biology, you often combine a phylogenetic tree (from one source) with trait data (from another source). If a species name is spelled slightly differently in the two datasets – for example, "Homo_sapiens" in the tree but "Homo_sapien" in your data – that species will be silently dropped from your analysis with no warning. This function catches those near-misses so you can fix them before running your analysis.

The function uses Levenshtein distance (the minimum number of single-character edits – insertions, deletions, or substitutions – needed to change one name into another) to measure similarity.

Value

A data frame with three columns:

name.in.data The name as it appears in your data.

name.in.tree The closest matching name on the tree.

differences Number of character differences (Levenshtein distance) between the two names.

Returns NULL invisibly if no close matches are found.

Examples

```
## Not run:
library(ape)
tree <- read.tree(text = "((Homo_sapiens, Pan_troglodytes), Gorilla_gorilla);")
my_data <- c("Homo_sapien", "Pan_troglodytes", "Gorila_gorilla")
FuzzyMatch(tree, my_data, max.dist = 2)
# Should flag "Homo_sapien" and "Gorila_gorilla" as close matches

## End(Not run)
```

getNe

Effective Population Size from Census Numbers

Description

Calculates the effective population size (N_e) from the number of breeding males and females. N_e is almost always smaller than the actual census size because not all individuals contribute equally to the next generation. Unequal sex ratios reduce N_e further.

Usage

```
getNe(males, females, locus = "A")
```

Arguments

males	Integer. Number of breeding males in the population.
females	Integer. Number of breeding females in the population.
locus	Character. Which locus type to calculate N_e for: "A" (default) – autosomal locus. "X" – X-linked locus.

Details

Two formulas are available:

- **Autosomal** (default): $N_e = 4MF/(M + F)$
- **X-linked**: $N_e = 9MF/(4M + 2F)$

where M = number of breeding males, F = number of breeding females.

When M = F, autosomal $N_e = 2N$ (same as census). As the sex ratio becomes more skewed, N_e drops. X-linked N_e differs because X chromosomes spend 2/3 of their time in females and 1/3 in males.

Value

Numeric. The effective population size.

Examples

```
# Equal sex ratio: 50 males, 50 females
getNe(50, 50)
# Returns 100 (same as census)

# Skewed: 10 males, 90 females (like elephant seals)
getNe(10, 90)
# Returns 36 -- much less than the census of 100!

# X-linked Ne with equal sex ratio
getNe(50, 50, locus = "X")
```

GetTipRates

Calculate Evolutionary Rates at Tree Tips

Description

Estimates how fast a discrete character has been evolving at each tip of a phylogenetic tree. For example, if you are studying chromosome number evolution, this tells you which species have recently experienced rapid changes in chromosome number on the branch leading to the present.

Usage

```
GetTipRates(
  tree = NULL,
  Q = NULL,
  tip.states = NULL,
  hyper = FALSE,
  p.mat = NULL
)
```

Arguments

<code>tree</code>	An ultrametric phylogenetic tree of class "phylo". "Ultrametric" means all tips are the same distance from the root (i.e., the tree represents time, not just branching pattern).
<code>Q</code>	A transition rate matrix (square matrix) with column names matching the possible character states. This matrix tells the model how likely each type of state change is. It is used internally for ancestral state reconstruction (estimating what states the ancestors had).
<code>tip.states</code>	An integer vector of observed states at the tips, with names matching the tree's tip labels. Values should be integers from 1 to N where N is the number of possible states. For example, if your trait has 3 states, each tip gets a value of 1, 2, or 3. If NULL, states will be derived from <code>p.mat</code> .
<code>hyper</code>	Logical. If TRUE, the model includes a binary hyperstate (state labels contain "h" which gets stripped during processing). This is an advanced option; most users should leave it as FALSE (the default).
<code>p.mat</code>	A probability matrix where rows are species (with row names matching tree tips) and columns are states. Each row should have exactly one 1 and the rest 0s (i.e., you know each species' state with certainty). This is an alternative way to provide tip states instead of <code>tip.states</code> .

Details

The "rate" for a tip is calculated as: $|tip_state - parent_state| / branch_length$. In plain English: how much the trait changed on the last branch of the tree (between the species and its most recent ancestor), divided by the length of that branch (a proxy for time). Bigger values mean faster recent evolution; a rate of zero means no change occurred on that branch.

The function works in three steps:

1. **Ancestral state reconstruction:** Uses `asr_mk_model` with the supplied transition matrix to estimate what state each internal node (ancestor) most likely had.
2. **Compare tip to parent:** For each species (tip), it looks at the reconstructed state of the immediately ancestral node (the parent node).
3. **Calculate rate:** The rate is the absolute difference between the tip state and its parent state, divided by the branch length connecting them.

Value

A named numeric vector of tip rates, one value per tip. Names correspond to tree tip labels. Higher values indicate faster evolution on that terminal branch.

Examples

```
## Not run:
library(ape)
library(castor)
tree <- rcoal(10)
# Make a simple 3-state transition matrix
```

```
Q <- matrix(c(-0.2, 0.1, 0.1,
              0.1, -0.2, 0.1,
              0.1, 0.1, -0.2), 3, 3)
colnames(Q) <- rownames(Q) <- c("1", "2", "3")
tip.states <- setNames(sample(1:3, 10, replace = TRUE), tree$tip.label)
rates <- GetTipRates(tree, Q, tip.states)
# Species with high rates evolved rapidly on their terminal branch

## End(Not run)
```

horn.beetle.csv	<i>Gnatocerus cornutus</i> measurements
-----------------	---

Description

A csv file containing measurements of horn and body size for the broad-horned flour beetle *Gnatocerus cornutus*. The file has three columns: individual ID, body size, and horn size. Used in the [ResSel](#) function examples for demonstrating residual selection analysis.

Examples

```
data <- read.csv(file = system.file("horn.beetle.csv", package = "evobiR"))
head(data)
```

hym.tree	<i>Hymenoptera phylogenetic tree</i>
----------	--------------------------------------

Description

A phylogenetic tree of class phylo containing 5 species of Hymenoptera (bees, wasps, and ants). Used in examples for the [FuzzyMatch](#) function to demonstrate fuzzy matching of taxon names between data and trees.

Examples

```
data(hym.tree)
ape::plot.phylo(hym.tree)
```

Description

An enhanced version of [make.simap](#) that adds configurable rejection limits and monitoring for large or complex datasets. Stochastic character mapping simulates possible histories of a discrete trait on a phylogeny – imagine painting each branch of the tree with colors representing different trait states.

Usage

```
make.simap2(
  tree,
  x,
  model = "SYM",
  nsim = 1,
  monitor = FALSE,
  rejmax = NULL,
  rejint = 1e+06,
  ...
)
```

Arguments

tree	A phylogenetic tree of class "phylo" or "multiPhylo".
x	A named vector of discrete character states at the tips, or a matrix of state probabilities (rows = tips, columns = states).
model	Character or matrix. Model of trait evolution: • "SYM" (default) – symmetric rates • "ER" – equal rates • "ARD" – all rates different • A custom rate matrix
nsim	Integer. Number of stochastic maps to simulate (default 1).
monitor	Logical. If TRUE, prints progress updates showing acceptance/rejection of each branch. Useful for debugging slow runs. Default is FALSE.
rejmax	Integer or NULL. Maximum number of rejection sampling attempts per branch before marking it as "fail." NULL (default) means no limit (keeps trying forever).
rejint	Integer. Interval for printing rejection count progress messages (default 1000000). Only used when monitor = TRUE.
...	Additional arguments passed to fitMk for rate estimation. Common options include: Q Transition rate matrix: "empirical" (default, estimated from the data), "mcmc" (sampled from posterior), or a fixed matrix. pi Root state priors: "equal", "estimated", "fitzjohn", or a numeric vector. message Logical. Print the Q matrix? Default TRUE.

Details

The key enhancement is the `rejmax` parameter: when the Gillespie simulation can't find a valid path on a branch after `rejmax` attempts, it marks that branch as "fail" instead of running forever. Failed branches can then be repaired with `fix.simmap`.

Value

A tree of class "simmap" (if `nsim = 1`) or a list of class `c("multiSimmap", "multiPhylo")` (if `nsim > 1`). Each tree has additional components:

maps A list of named vectors, one per edge, giving the time spent in each state along that edge.

mapped.edge A matrix summarizing total time in each state per edge.

Q The transition rate matrix used.

logL The log-likelihood.

fail Integer vector of edge indices that failed (hit `rejmax`).

See Also

`fix.simmap` to repair failed branches.

Examples

```
## Not run:
library(phytools)
# Basic usage (same as make.simmap)
mapped <- make.simmap2(tree, states, model = "ARD", nsim = 100)

# With rejection limit and monitoring
mapped <- make.simmap2(tree, states, model = "ARD", nsim = 10,
                      rejmax = 500000, monitor = TRUE)
# Fix any failed branches
fixed <- fix.simmap(mapped, tip_data, trans_matrix)

## End(Not run)
```

mite.trait

Phenotype data for mites

Description

A dataframe of sexual system and chromosome number data for mites. Contains three columns:

Column 1 species names matching the tip labels of `trees.mite`

Column 2 diploid chromosome numbers (continuous trait)

Column 3 sex determination system coded as 0 (diplodiploidy) or 1 (haplodiploidy) (discrete binary trait)

Used in examples for the `AncCond` function.

Examples

```
data(mite.trait)
head(mite.trait)
```

Mode

Statistical Mode

Description

Returns the most frequently occurring value in a vector. R does not have a built-in mode function for data (its `mode()` returns storage type), so this simple helper fills that gap.

Usage

```
Mode(x)
```

Arguments

`x` A vector of any type.

Value

The most common value in `x`. If there is a tie, the first value (in order of appearance) wins.

Examples

```
Mode(c(1, 2, 2, 3))
```

`plot.phyloscaled`
Plot a Rate-Scaled Phylogeny

Description

S3 plot method for "phyloscaled" objects produced by [scaleTreeRates](#). Visualizes how evolutionary rates vary across the tree using either branch length scaling or color coding.

Usage

```
## S3 method for class 'phyloscaled'
plot(
  x,
  method = "multiply",
  palette = "RdYlGn",
  edge.width = 1,
  cex = 1,
  show.tip.label = TRUE,
  ...
)
```

Arguments

x	A tree of class "phylocaled" (output of scaleTreeRates). Contains a \$scalar element indicating the rate multiplier for each edge.
method	Character. How to display rate variation: • "multiply" (default) – multiply each edge length by its scalar, so fast branches appear longer and slow branches appear shorter. • "color" – color edges by their scalar value. Red/cool colors = slow evolution (scalar < 1), green/warm colors = fast evolution (scalar > 1).
palette	Character. Color palette for the "color" method. Default is "RdYlGn" (from RColorBrewer). Automatically switches to "viridis" if there are more than 10 categories.
edge.width	Numeric. Width of plotted edges (default 1).
cex	Numeric. Character expansion factor for tip labels (default 1).
show.tip.label	Logical. Whether to display tip labels (default TRUE).
...	Additional arguments passed to plot.phylo .

Value

Produces a plot. Returns NULL invisibly.

Examples

```
## Not run:
# After running scaleTreeRates:
scaled_tree <- scaleTreeRates(tree, tip.states, model = "ARD")

# Method 1: scale branch lengths
plot(scaled_tree, method = "multiply")

# Method 2: color branches by rate
plot(scaled_tree, method = "color")

## End(Not run)
```

Description

Computes the expected proportion of chromosomal fusions that join a sex chromosome to an autosome (SA-fusions), as opposed to autosome-autosome (AA) or sex-sex (SS) fusions. Two null models are available:

Usage

```
pSAF(n_auto, n_x = 0, n_y = 0, n_z = 0, n_w = 0, mu = 0.5, weighted = TRUE)
```

Arguments

n_auto	Integer ≥ 1 . The haploid autosome count — i.e., the number of autosome PAIRS. For example, humans have 22 autosome pairs, so $n_auto = 22$. The diploid autosome count used internally is $2 * n_auto$. Can be a vector to compute probabilities across a range of autosome counts simultaneously.
n_x	Integer ≥ 0 . Number of X chromosomes present in the heterogametic sex (males in XY systems). For a standard XY system, $n_x = 1$. For an XXY system, $n_x = 2$. For an XO system, $n_x = 1$ and $n_y = 0$. Must not be used together with n_z or n_w .
n_y	Integer ≥ 0 . Number of Y chromosomes present in the heterogametic sex. For a standard XY system, $n_y = 1$. For an XYY system, $n_y = 2$. Set to 0 for XO systems. Must not be used together with n_z or n_w .
n_z	Integer ≥ 0 . Number of Z chromosomes present in the heterogametic sex (females in ZW systems). Analogous to n_x but for ZW systems. Must not be used together with n_x or n_y .
n_w	Integer ≥ 0 . Number of W chromosomes present in the heterogametic sex. Analogous to n_y but for ZW systems. Must not be used together with n_x or n_y .
mu	Numeric between 0 and 1. The proportion of fusion mutations that originate in females , regardless of which sex is homogametic. Default is 0.5, meaning both sexes contribute equally. Setting $mu = 0.7$ always means 70% of fusions arise in females — in an XY system that is the homogametic sex, but in a ZW system it is the heterogametic sex. The function handles this mapping internally so the user can always think in terms of phenotypic sex.
weighted	Logical. If TRUE (default), returns the fixation- weighted probability of SA-fusions (Schmalz & Blackmon). If FALSE, returns the occurrence probability only (Anderson et al. 2020).

Details

Occurrence model ($weighted = FALSE$): The original null from Anderson et al. (2020). Every pair of non-homologous chromosomes is equally likely to fuse. The probability of each fusion class (AA, SA, SS) depends only on how many chromosomes of each type are present in each sex, weighted by the proportion of fusions originating in each sex (mu).

Fixation-weighted model ($weighted = TRUE$, default): Extends the occurrence model by accounting for the fact that a new fusion's probability of drifting to fixation under neutrality depends on how many copies of that chromosome exist in the population (Kimura 1962). Autosomes are present in $2N$ copies, the shared sex chromosome (X or Z) in $3N/2$ copies, and the limited sex chromosome (Y or W) in only $N/2$ copies. Because fewer copies means higher fixation probability, Y-autosome (or W-autosome) fusions are disproportionately likely to reach fixation relative to their occurrence rate. This model therefore predicts a higher baseline proportion of SA-fusions than the occurrence model alone.

Terminology:

- *Shared sex chromosome*: the sex chromosome present in both sexes — X in XY systems (females are XX, males have at least one X) or Z in ZW systems (males are ZZ, females have at least one Z).
- *Limited sex chromosome*: the sex chromosome present only in the heterogametic sex — Y in XY systems or W in ZW systems.
- *Homogametic sex*: the sex with two copies of the shared sex chromosome (e.g., XX females in XY systems, ZZ males in ZW systems).
- *Heterogametic sex*: the sex carrying both shared and limited sex chromosomes (e.g., XY males, ZW females).
- *Non-homologous fusion*: a fusion between two chromosomes that are NOT homologs of each other. Fusions between homologs (e.g., both copies of chromosome 3) produce unbalanced gametes and are excluded.

How the math works:

Within each sex, the probability that a random non-homologous fusion falls into each class (AA, SA, SS) is determined by the number of chromosomes of each type. The overall probability is the weighted average across the two sexes (using μ).

For the fixation-weighted model, the SA class is further split into fusions involving the shared sex chromosome (XA or ZA) versus the limited sex chromosome (YA or WA). Each class is multiplied by a fixation weight proportional to $1/(\text{population copy number})$, then renormalized so probabilities sum to 1.

Fixation weights (relative to autosomes):

- Autosome: weight = 1 (present in $2N$ copies)
- Shared sex chrom (X/Z): weight = $4/3$ (present in $3N/2$ copies)
- Limited sex chrom (Y/W): weight = 4 (present in $N/2$ copies)

Value

A numeric value (or vector, if `n_auto` is a vector) giving the probability that a random fusion is a sex-autosome fusion under the chosen null model.

References

- Anderson, N.W., Hjelmen, C.E., & Blackmon, H. (2020). The probability of fusions joining sex chromosomes and autosomes. *Biology Letters*, 16, 20200648. doi:10.1098/rsbl.2020.0648
- Schmalz, S. & Blackmon, H. A fixation-weighted null model for the proportion of sex chromosome-autosome fusions. *In prep.*
- Kimura, M. (1962). On the probability of fixation of mutant genes in a population. *Genetics*, 47, 713-719.

Examples

```
# ---- Basic usage ----

# Standard XY system with 5 autosome pairs (Da = 10), fixation-weighted
pSAF(n_auto = 5, n_x = 1, n_y = 1)
```

```

# Same system, occurrence probability only (Anderson et al. 2020)
pSAF(n_auto = 5, n_x = 1, n_y = 1, weighted = FALSE)

# ---- Different sex chromosome systems ----

# XO system (no Y chromosome): 13 autosome pairs
pSAF(n_auto = 13, n_x = 1, n_y = 0)

# XXY system: 2 X chromosomes and 1 Y in the heterogametic sex
pSAF(n_auto = 5, n_x = 2, n_y = 1)

# XYY system: 1 X and 2 Y chromosomes in the heterogametic sex
pSAF(n_auto = 5, n_x = 1, n_y = 2)

# ---- ZW systems ----

# Standard ZW (e.g., birds): 10 autosome pairs
pSAF(n_auto = 10, n_z = 1, n_w = 1)

# ---- Vectorized over autosome count ----

# Compute across a range of autosome counts for plotting
n <- 1:30
p_fix <- pSAF(n_auto = n, n_x = 1, n_y = 1)
p_occ <- pSAF(n_auto = n, n_x = 1, n_y = 1, weighted = FALSE)
plot(n, p_fix, type = "l", col = "red",
      xlab = "Haploid autosome count", ylab = "P(SA fusion)")
lines(n, p_occ, col = "blue", lty = 2)
legend("topright", c("Fixation-weighted", "Occurrence"),
      col = c("red", "blue"), lty = 1:2)

# ---- Habronattus example from the paper (XX0, Da = 26) ----
pSAF(n_auto = 13, n_x = 2, n_y = 0) # fixation: 0.243
pSAF(n_auto = 13, n_x = 2, n_y = 0, weighted = FALSE) # occurrence: 0.194

```

ReOrderAlignment

Reorder Sequences in a FASTA Alignment

Description

Reads a FASTA alignment and writes a new file with the sequences reordered by their starting position (i.e., where the first non-gap character appears).

Usage

```
ReOrderAlignment(file, newfile, ref = NULL)
```

Arguments

file	Character. Path to the input FASTA alignment file.
newfile	Character. Path for the output (reordered) FASTA file.
ref	Optional. Reference sequence(s) to place first in the output. Can be specified as: • A character vector of sequence names. • A numeric vector of sequence indices (positions in the file). • NULL (default) – no pinned references.

Details

When would you use this? When you have an alignment where different sequences cover different regions – for example, amplicon sequencing data where each primer pair targets a different part of a gene, or partial genome sequences that start at different positions. Sorting by start position makes the alignment much easier to visualize and inspect in an alignment viewer.

Optionally, one or more reference sequences can be pinned to the top of the output file regardless of their start position.

Value

Writes the reordered alignment to newfile. No value is returned (called for its side effect of writing a file).

Examples

```
## Not run:
# Reorder by start position
ReOrderAlignment("my_alignment.fasta", "reordered.fasta")

# Pin "reference_seq" to the top, then sort the rest
ReOrderAlignment("my_alignment.fasta", "reordered.fasta",
  ref = "reference_seq")

## End(Not run)
```

ResSel

Residual Selection for Artificial Breeding Experiments

Description

Identifies individuals with the most extreme residual values from a linear regression between two traits. This is a common method in artificial selection experiments.

Usage

```
ResSel(data, traits, percent = 10, identifier = 1, model = "linear")
```

Arguments

<code>data</code>	A data frame containing at least an identifier column and the two trait columns.
<code>traits</code>	A numeric vector of length 2 giving the column numbers of the two traits. The first is the predictor (x-axis), the second is the response (y-axis). For example, <code>c(2, 3)</code> means column 2 predicts column 3.
<code>percent</code>	Numeric. The percentage of individuals to select from each tail of the residual distribution (default 10, meaning the top 10% and bottom 10%). Higher values select more individuals but with less extreme phenotypes.
<code>identifier</code>	Integer. Column number containing individual IDs (default 1).
<code>model</code>	Character. Type of regression model (default "linear"). Currently only linear models are supported.

Details

What are residuals? Imagine you plot body size (x-axis) against wing length (y-axis) and draw a best-fit line through the data. A "residual" is how far above or below that line each individual falls. Individuals far ABOVE the line have unusually long wings for their body size (positive residuals). Individuals far BELOW the line have unusually short wings for their body size (negative residuals). The function selects the most extreme individuals from each end: the "high line" (furthest above the line) and the "low line" (furthest below the line) for use as breeding stock. It also produces a scatter plot showing the regression line (orange), high-line individuals (blue), and low-line individuals (red).

Value

A list with two elements:

high line Identifiers of individuals in the upper tail (positive residuals – their y-trait is bigger than expected given their x-trait value).

low line Identifiers of individuals in the lower tail (negative residuals – their y-trait is smaller than expected given their x-trait value).

Examples

```
## Not run:
# Simulate some data: 100 beetles with body size and wing length
df <- data.frame(
  id = paste0("beetle_", 1:100),
  body_size = rnorm(100, 50, 10),
  wing_length = rnorm(100, 30, 5)
)
# Select top and bottom 15% by residual wing length
# (controlling for body size)
selected <- ResSel(df, traits = c(2, 3), percent = 15)
selected$`high line` # beetles with unusually LONG wings for their size
selected$`low line`  # beetles with unusually SHORT wings for their size

## End(Not run)
```

SampleTrees

Sample Trees from Bayesian MCMC Output

Description

Reads a collection of phylogenetic trees from a NEXUS file (typically output from a Bayesian analysis like MrBayes or BEAST), removes the burn-in, and randomly samples a specified number of post-burn-in trees.

Usage

```
SampleTrees(trees, burnin, final.number, format, prefix)
```

Arguments

trees	Character. Path to a NEXUS format file containing phylogenetic trees (e.g., the .t file from MrBayes or the .trees file from BEAST).
burnin	Numeric between 0 and 1. Proportion of trees to discard as burn-in. For example, 0.25 removes the first 25% of trees. A common starting point is 0.25, but you should check convergence in Tracer or a similar tool to choose wisely.
final.number	Integer. Number of trees to randomly sample from the post-burn-in trees. Common choices are 100 or 1000.
format	Character. Output format: • "new" – Newick format (.nwk file) • "nex" – Nexus format (.nex file)
prefix	Character. Prefix for the output file name. The appropriate extension (.nwk or .nex) is added automatically.

Details

Why do you need this? Bayesian phylogenetic programs like MrBayes produce thousands (sometimes millions) of trees as they explore "tree space." The first chunk of trees are unreliable because the program was still searching for good trees – this early period is called the "burn-in" and should be discarded. After removing the burn-in, you typically don't need ALL the remaining trees (they would be too many to work with), so you randomly subsample a manageable number that still represents the full range of uncertainty in your phylogeny.

Value

Writes the sampled trees to a file. No value is returned (called for its side effect of writing a file).

Examples

```
## Not run:
# Your MrBayes run produced "my_analysis.nex.t" with 10,000 trees.
# Remove the first 25% as burn-in, then randomly pick 100 trees:
SampleTrees("my_analysis.nex.t", burnin = 0.25,
            final.number = 100, format = "new",
            prefix = "sampled_trees")
# This creates "sampled_trees.nwk" with 100 trees

## End(Not run)
```

scaleTreeRates

Analyze Rate Heterogeneity in Discrete Character Evolution

Description

Tests whether different branches of a phylogeny have different rates of discrete character evolution. The function assigns a "scalar" multiplier to each edge: scalar > 1 means faster-than-average evolution, scalar < 1 means slower, and scalar = 1 means the background rate.

Usage

```
scaleTreeRates(
  tree,
  tip.states,
  model,
  fixedQ = NULL,
  max.ratio = 2,
  nbins = 10,
  max.transition = 1,
  var.start = FALSE,
  pi = "fitzjohn"
)
```

Arguments

- | | |
|------------|---|
| tree | A phylogenetic tree of class "phylo". |
| tip.states | A named vector of discrete character states at the tips. Names must match tree\$tip.label. |
| model | Character or matrix. The model for fitMk : <ul style="list-style-type: none"> • "ER" – equal rates • "SYM" – symmetric rates • "ARD" – all rates different • A custom rate index matrix |

<code>fixedQ</code>	A pre-estimated Q (transition rate) matrix. If NULL (default), rates are estimated from the data.
<code>max.ratio</code>	Numeric. Maximum rate scalar to test (default 2). The function tests scalars from $1/\text{max.ratio}$ to <code>max.ratio</code> .
<code>nbins</code>	Integer. Number of scalar bins above and below 1 (default 10). More bins gives finer resolution but takes longer.
<code>max.transition</code>	Integer. Maximum number of bin steps an edge's scalar can differ from its parent's scalar (default 1). This prevents rate estimates from jumping wildly between adjacent branches.
<code>var.start</code>	Logical. If TRUE, tries multiple starting scalar values at the root and returns the best tree. If FALSE (default), fixes the root scalar at 1.
<code>pi</code>	Character. Method for estimating the root state prior. Passed to <code>fitMk</code> . Default is "fitzjohn".

Details

The method works by trying different scalar values for each edge (via dynamic programming) and keeping whichever value maximizes the likelihood.

The algorithm proceeds edge-by-edge from root to tips. For each edge, it tries every allowed scalar value (constrained to be within `max.transition` bins of the parent's scalar), fits the Mk model with scaled branch lengths, and picks the scalar that gives the highest likelihood.

Value

A tree of class `c("phyloscaled", "phylo")` with an additional `$scalar` element: a numeric vector (one value per edge) of rate multipliers. Plot with `plot.phyloscaled`.

Examples

```
## Not run:
library(phytools)
tree <- rcoal(20)
states <- setNames(sample(1:3, 20, replace = TRUE), tree$tip.label)
scaled <- scaleTreeRates(tree, states, model = "ARD", nbins = 5)
plot(scaled, method = "color")

## End(Not run)
```

Description

Applies a function across a sliding window over a vector or matrix.

Usage

```
SlidingWindow(FUN, data, window, step, strict = FALSE)
```

Arguments

<code>FUN</code>	Function to apply within each window. Must accept a vector (for vector data) or matrix (for matrix data) and return a single value. Can be a function name as a string (e.g., "mean") or a function object.
<code>data</code>	A numeric vector or matrix to analyze.
<code>window</code>	Integer. Size of the sliding window (number of elements for vectors, or number of rows/columns for matrices).
<code>step</code>	Integer. How far the window moves between calculations. A step of 1 means windows overlap maximally; a step equal to window means no overlap.
<code>strict</code>	Logical. If TRUE, validates that data is numeric and that window/step are sensible sizes. Default is FALSE.

Details

What is a sliding window? Imagine looking at a long DNA sequence through a small window that you slide along one step at a time. At each position, you measure something inside the window (e.g., the average GC content). This gives you a profile showing how that measurement changes along the sequence. For example, you could calculate the mean GC content in 100-bp windows sliding every 10 bp across a genome, or compute local diversity statistics across a landscape grid.

For vector data the window slides along the length; for matrix data the window slides in both dimensions (rows and columns).

Value

For vector input: a numeric vector of results, one per window position.

For matrix input: a matrix of results where each cell is the function applied to the corresponding window.

Examples

```
# Calculate mean in windows of 10, stepping by 5
x <- rnorm(100)
SlidingWindow("mean", x, window = 10, step = 5)

# GC content in sliding windows (simplified example)
seq_data <- sample(c(0, 1), 500, replace = TRUE)
gc <- SlidingWindow("mean", seq_data, window = 50, step = 10)
```

Description

Reads multiple sequence alignment files and either concatenates them into a single supermatrix or pads each one to include all taxa.

Usage

```
SuperMatrix(
  missing = "-",
  prefix = "concatenated",
  save = TRUE,
  input = "",
  format.in = "f",
  format.out = "f",
  concatenate = TRUE
)
```

Arguments

missing	Character. Symbol to use for missing data (default "-").
prefix	Character. Prefix for output file names (default "concatenated").
save	Logical. If TRUE (default), saves the supermatrix FASTA and partition CSV to disk.
input	Character. A pattern to match alignment file names in the current directory. If "" (default), all files are used. For example, input = ".fasta" would only read files with ".fasta" in their name.
format.in	Character. Input alignment format (default "f" for FASTA). Passed to read.dna .
format.out	Character. Output alignment format (default "f" for FASTA). Passed to write.dna .
concatenate	Logical. If TRUE (default), concatenates all genes into a single supermatrix. If FALSE, produces separate padded alignments (each containing all taxa).

Details

What is a supermatrix? Think of it like merging spreadsheets. You have separate DNA sequence files for different genes (e.g., gene1.fasta with 80 species, gene2.fasta with 95 species). This function pastes all the gene columns together side-by-side into one big alignment. Species that are missing from a particular gene get filled in with the missing data character (default "-").

A partition file is also saved so you know which columns in the final alignment came from which gene – this is important for partitioned phylogenetic analyses where each gene gets its own substitution model.

Value

When concatenate = TRUE: a list with two elements:

partitions A data frame with columns: gene file name, start position, stop position.

supermatrix A character matrix of the concatenated alignment.

When concatenate = FALSE: a list of padded alignment matrices, one per input file.

Examples

```
## Not run:
# Concatenate all .fasta files in the current directory
setwd("path/to/alignments")
result <- SuperMatrix(input = ".fasta")
result$partitions # see which positions belong to which gene

## End(Not run)
```

trees.mite	<i>10 phylogenetic trees for mites</i>
------------	--

Description

A multiPhylo object containing 10 phylogenetic trees from a previously published study on mite sexual system evolution. These trees can be used with the associated [mite.trait](#) dataset for comparative analyses using [AncCond](#). Loaded as the object trees.

Examples

```
data(trees.mite)
plot(trees[[1]])
```

trees.nex	<i>100 simulated phylogenetic trees</i>
-----------	---

Description

A nexus-format file containing a collection of 100 simulated phylogenetic trees, each with 10 tips. Stored in the `inst/` directory. Used in examples for the [SampleTrees](#) function to demonstrate tree subsampling from posterior distributions.

WinCalcD

*Windowed Patterson's D-Statistic***Description**

Calculates Patterson's D-statistic in sliding windows across a sequence alignment. This lets you see how introgression signal varies along the genome – some regions may show strong introgression while others don't.

Usage

```
WinCalcD(
  alignment = "alignment.fasta",
  win.size = 100,
  step.size = 50,
  boot = FALSE,
  replicate = 1000
)
```

Arguments

<code>alignment</code>	Character. Path to the FASTA alignment file (default "alignment.fasta").
<code>win.size</code>	Integer. Width of each window in base pairs (default 100).
<code>step.size</code>	Integer. How far the window slides between calculations (default 50). Setting <code>step.size < win.size</code> gives overlapping windows for smoother results.
<code>boot</code>	Logical. If TRUE, performs bootstrap significance testing within each window. Default is FALSE.
<code>replicate</code>	Integer. Number of bootstrap replicates per window (default 1000). Only used when <code>boot = TRUE</code> .

Details

Like [CalcD](#), the alignment must have exactly 4 sequences in the order P1, P2, P3, Outgroup.

D is calculated as $(ABBA - BABA) / (ABBA + BABA)$ within each window. Windows with no informative sites return $D = 0$. When bootstrapping, sites within each window are resampled with replacement to generate a null distribution.

Value

A data frame with one row per window and columns:

range Genomic coordinates of the window (e.g., "1:100").

abba Number of ABBA sites in the window.

baba Number of BABA sites in the window.

d D-statistic for the window.

Z Z-score (only if `boot = TRUE`).

pval P-value (only if `boot = TRUE`).

Examples

```
## Not run:
# Calculate D in 500-bp windows sliding every 250 bp
results <- WinCalcD("my_alignment.fasta",
                    win.size = 500, step.size = 250)

# Plot D across the genome
plot(1:nrow(results), as.numeric(results$d), type = "l",
     xlab = "Window", ylab = "D-statistic")
abline(h = 0, lty = 2, col = "red")

## End(Not run)
```

Index

* comparative method

AncCond, 4

* continuous trait

AncCond, 4

* datasets

1.fasta, 3

horn.beetle.csv, 16

hym.tree, 16

mite.trait, 18

trees.mite, 31

trees.nex, 31

* discrete trait

AncCond, 4

1.fasta, 3

3.fasta (1.fasta), 3

AncCond, 4, 18, 31

asr_mk_model, 15

CalcD, 3, 6, 8, 32

CalcPopD, 3, 8

countTrees, 9

evobiR (evobiR-package), 2

evobiR-package, 2

fitMk, 17, 27, 28

fix.simap, 11, 18

FuzzyMatch, 12, 16

getNe, 13

GetTipRates, 14

horn.beetle.csv, 16

hym.tree, 16

make.simap, 11, 17

make.simap2, 11, 17

mite.trait, 18, 31

Mode, 19

plot.phylo, 20

plot.phyloscaled, 19, 28

pSAF, 20

read.alignment, 7, 8

read.dna, 30

ReOrderAlignment, 23

ResSel, 16, 24

SampleTrees, 26, 31

scaleTreeRates, 19, 20, 27

SlidingWindow, 28

SuperMatrix, 30

trees (trees.mite), 31

trees.mite, 18, 31

trees.nex, 31

WinCalcD, 3, 32

write.dna, 30