Package 'bifrost'

November 5, 2026

Title Branch-Level Inference Framework for Recognizing Optimal Shifts in Traits

Version 0.2.3

Description Methods for detecting, visualizing, and evaluating cladogenic shifts in multivariate trait data on phylogenies. Implements penalized-likelihood multivariate generalized least squares models and a greedy step-wise shift search for high-dimensional trait datasets and large trees via searchOptimalConfiguration(). Provides tools for inspecting search trajectories, summarizing branch and lineage rates, analyzing shift timing and magnitudes, estimating post-hoc regime covariance and integration, and running simulation-based calibration and tuning. The search follows approaches developed in Smith et al. (2023) <doi:10.1111/nph.19099> and Berv et al. (2024) <doi:10.1126/sciadv.adp0114>. Methods build on multivariate generalized least squares approaches described in Clavel et al. (2019) <doi:10.1093/sysbio/syy045> and implemented in the mvgls() function from the 'mvMORPH' package. Documentation and worked examples are available at <https://jakeberv.com/bifrost/>.

License GPL (>= 2)

Encoding UTF-8

URL https://jakeberv.com/bifrost/, https://github.com/jakeberv/bifrost

BugReports https://github.com/jakeberv/bifrost/issues

Depends R (>= 4.2)

Imports ape, cli (>= 3.6.0), corpcor, digest, parallelly, future (>= 1.49.0), future.apply, progressr, RhpcBLASctl, phytools (>= 2.0-3), plotrix, RRphylo, grDevices, graphics, grid, jsonlite, stats, mvMORPH, viridis, txtplot

Suggests ComplexHeatmap, RColorBrewer, circlize, evd, phylolm, rmarkdown, testthat (>= 3.0.0), univariateML, withr, spelling

Config/Needs/website classInt, geomorph, ggplot2, htmltools, knitr, patchwork, pkgload, plotly, png, scatterplot3d, yaml

Config/testthat/edition 3

Config/testthat/parallel false

Config/build/copy-method copy

Language en-US

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Jacob S. Berv [aut, cre, cph, fnd] (ORCID:
<https://orcid.org/0000-0002-5962-0621>),
Nathan Fox [aut] (ORCID: <https://orcid.org/0000-0002-2816-9751>),
Matt J. Thorstensen [aut] (ORCID:
<https://orcid.org/0000-0002-7870-3369>),
Henry Lloyd-Laney [aut] (ORCID:
<https://orcid.org/0000-0003-4650-8937>),
Emily M. Troyer [aut] (ORCID: <https://orcid.org/0000-0001-7478-2306>),
Rafael A. Rivero-Vega [aut] (ORCID:
<https://orcid.org/0000-0001-5937-6377>),
Stephen A. Smith [aut, fnd] (ORCID:
<https://orcid.org/0000-0003-2035-9531>),
Matt Friedman [aut, fnd] (ORCID:
<https://orcid.org/0000-0002-0114-7384>),
David F. Fouhey [aut, fnd] (ORCID:
<https://orcid.org/0000-0001-5028-5161>),
Brian C. Weeks [aut, fnd] (ORCID:
<https://orcid.org/0000-0003-2967-2970>)

Maintainer Jacob S. Berv <jacob.berv@gmail.com>

Repository CRAN

Date/Publication 2026-11-05 03:30:02 UTC

Contents

as.data.frame.bifrost_search_tuning_grid 4
as.data.frame.bifrost_search_tuning_selection 4
as.data.frame.bifrost_shift_recovery_evaluation 5
as.data.frame.bifrost_simulation_study 6
as.data.frame.regime_correlation_pca 6
as.data.frame.regime_integration_relationships 7
as.data.frame.regime_module_diagnostics 8
bifrost_example_file 9
bootstrap_rate_distribution 9
compare_shift_magnitudes 11
createSimulationTemplate 13
distribution-fit-data-frames 15
evaluateShiftRecovery 17
fisher_z_transform 19
fit_rate_distribution 20
fit_regime_covariances 21
fit_regime_covariance_runs 23
fit_waiting_time_distribution 24
generateViridisColorScale 26

icTrajectory . . . 27
lineage_rates . . . 29
plot.icTrajectory . . . 34
plot.rateMap . . . 36
plot.rate_distribution_fit . . . 40
plot.regime_correlation_pca . . . 41
plot.regime_integration_relationships . . . 42
plot.regime_module_diagnostics . . . 44
plot.shift_magnitude_comparison . . . 45
plot.shift_magnitude_comparison_set . . . 47
plot.shift_magnitude_counts . . . 47
plot.shift_magnitude_count_set . . . 49
print.bifrost_search . . . 49
print.bifrost_search_tuning_grid . . . 50
print.bifrost_search_tuning_selection . . . 50
print.bifrost_shift_recovery_evaluation . . . 51
print.bifrost_simulation_study . . . 51
print.bifrost_simulation_template . . . 52
print.rateMap . . . 53
rateMap . . . 54
rateMapControl . . . 61
rateMapRateFlags . . . 63
rateMapView . . . 65
regime_correlation_pca . . . 66
regime_integration_pgls . . . 68
regime_integration_relationships . . . 70
regime_module_diagnostics . . . 71
runFalsePositiveSimulationStudy . . . 72
runSearchTuningGrid . . . 75
runShiftRecoverySimulationStudy . . . 79
searchOptimalConfiguration . . . 82
selectTunedSearchParameters . . . 89
shift_magnitude_counts . . . 92
shift_magnitude_groups . . . 93
shift_node_marks . . . 94
shift_transitions . . . 97
shift_waiting_times . . . 98
simulateNullDataset . . . 99
simulateShiftedDataset . . . 101
summarize_regime_covariances . . . 104
summarize_regime_covariance_runs . . . 106

Index **108**

as.data.frame.bifrost_search_tuning_grid

Convert a fixed-IC search tuning grid to a data frame

Description

Convert a fixed-IC search tuning grid to a data frame

Usage

```
## S3 method for class 'bifrost_search_tuning_grid'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A bifrost_search_tuning_grid object.
row.names, optional	Arguments required by the S3 generic; ignored.
...	Unused.

Value

The summary_table component as a data frame.

as.data.frame.bifrost_search_tuning_selection

Convert a tuned search-parameter selection to a data frame

Description

Convert a tuned search-parameter selection to a data frame

Usage

```
## S3 method for class 'bifrost_search_tuning_selection'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("selected", "feasible"),
  ...
)
```

Arguments

x	A bifrost_search_tuning_selection object.
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which table to return: "selected" for the chosen row or "feasible" for the filtered/ranked table.
...	Unused.

Value

A data frame for the requested selection component.

---

as.data.frame.bifrost_shift_recovery_evaluation

Convert a shift-recovery evaluation to a data frame

Description

Convert a shift-recovery evaluation to a data frame

Usage

```
## S3 method for class 'bifrost_shift_recovery_evaluation'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("metrics", "counts"),
  ...
)
```

Arguments

x	A bifrost_shift_recovery_evaluation object.
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which table to return: "metrics" for performance metrics or "counts" for aggregated contingency counts.
...	Unused.

Value

A data frame for the requested evaluation component.

as.data.frame.bifrost_simulation_study

Convert a bifrost simulation study to a data frame

Description

Convert a bifrost simulation study to a data frame

Usage

```
## S3 method for class 'bifrost_simulation_study'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("per_replicate", "summary"),
  ...
)
```

Arguments

x	A bifrost_simulation_study object.
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which table to return: "per_replicate" for the main replicate table or "summary" for a one-row study summary.
...	Unused.

Value

A data frame for the requested study component.

as.data.frame.regime_correlation_pca

Convert Regime Correlation PCA Results to Data Frames

Description

Convert Regime Correlation PCA Results to Data Frames

Usage

```
## S3 method for class 'regime_correlation_pca'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("scores", "loadings", "variance", "diagnostics"),
  ...
)
```

Arguments

x	A regime_correlation_pca object from regime_correlation_pca().
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which component to return: "scores", "loadings", "variance", or "diagnostics".
...	Unused.

Value

A data frame for the requested PCA component.

---

as.data.frame.regime_integration_relationships

Convert Regime Integration Relationship Results to Data Frames

---

Description

Convert Regime Integration Relationship Results to Data Frames

Usage

```
## S3 method for class 'regime_integration_relationships'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
 component = c("combined", "variance_points", "correlation_points", "variance_removed",
    "correlation_removed", "variance_curve", "correlation_curve", "models"),
  ...
)
```

Arguments

x	A regime_integration_relationships object.
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which component to return.
...	Unused.

Value

A data frame for the requested component.

---

as.data.frame.regime_module_diagnostics *Convert Regime Module Diagnostics to Data Frames*

Description

Convert Regime Module Diagnostics to Data Frames

Usage

```
## S3 method for class 'regime_module_diagnostics'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("scores", "correlations", "plot_data", "comparisons"),
  ...
)
```

Arguments

x	A regime_module_diagnostics object.
row.names, optional	Arguments required by the S3 generic; ignored.
component	Which component to return: "scores", "correlations", "plot_data", or "comparisons".
...	Unused.

Value

A data frame for the requested component.

bifrost_example_file *Locate a verified empirical example-data artifact*

Description

Resolves a named empirical artifact to a locally verified file. A directory supplied through BIFROST_ARTIFACT_DIR always takes precedence, including when refresh = TRUE. With no verified cache entry, the first request uses the manifest and artifact currently tracked on GitHub main. Later calls reuse the verified cache and work offline; use refresh = TRUE to check main again. The returned value is a path; this function does not deserialize the file.

Usage

```
bifrost_example_file(name, refresh = FALSE, quiet = FALSE)
```

Arguments

name	A registered example-data identifier.
refresh	Whether to bypass the cached manifest and check the remote manifest for a newer artifact.
quiet	Whether downloads should suppress progress output.

Value

A normalized path to the checksum-verified artifact.

Examples

```
## Not run:
bifrost_example_file("jaw-tree")
bifrost_example_file("jaw-tree", refresh = TRUE)

## End(Not run)
```

bootstrap_rate_distribution *Bootstrap A Fitted Rate Distribution*

Description

Precompute parametric bootstrap parameter draws for a fitted rate distribution. This keeps stochastic uncertainty estimation out of the plotting method while preserving the bootstrap draws on the returned rate_distribution_fit object.

Usage

```
bootstrap_rate_distribution(
  x,
  model = NULL,
  reps = 1000L,
  seed = NULL,
  kl = NULL
)
```

Arguments

x	A rate_distribution_fit object from fit_rate_distribution().
model	Distribution model to bootstrap. If NULL, the selected model stored in x$selected_model is used.
reps	Number of bootstrap parameter draws.
seed	Optional random seed for the bootstrap draw. The previous global RNG state is restored after the bootstrap when a seed is supplied.
kl	NULL or logical. When NULL, compute the bootstrap Kullback-Leibler divergence summary for Gumbel fits and skip it for other models. Set to FALSE to store only the bootstrap parameter draws.

Details

For Gumbel fits, the default kl = NULL also computes the Kullback-Leibler divergence between the empirical kernel density of the fitted data and the fitted Gumbel density, then repeats that calculation for each bootstrap parameter draw. Extract the compact diagnostic table with as.data.frame(x, component = "goodness").

Value

A rate_distribution_fit object with a rate_distribution_bootstrap entry stored in x$bootstrap[[model]]. The entry contains the model name, number of draws, seed, and parameter matrix returned by univariateML::bootstrapml(). For Gumbel fits, the entry also stores a bootstrapped Kullback-Leibler divergence summary unless kl = FALSE.

Examples

```
if (requireNamespace("univariateML", quietly = TRUE) &&
    requireNamespace("evd", quietly = TRUE)) {
  fit <- fit_rate_distribution(c(1, 1.4, 1.9, 2.8, 4.1, 6.5), models = "gumbel")
  fit <- bootstrap_rate_distribution(fit, model = "gumbel", reps = 25, seed = 1)
}
```

compare_shift_magnitudes *Compare Rate-Shift Magnitudes*

Description

Compare the magnitude of fitted rate increases and decreases from a `bifrost_search` BMM result or a transition table returned by `shift_transitions()`. This is the package-level analogue of the Berv et al. (2026) workflow that summarized rate changes and tested whether increase and decrease magnitudes differed. When `x` is a `bifrost_search` object, it must be a multi-regime BMM fit; use a transition table or `tree` plus `state_values` for generic mapped-tree inputs.

Usage

```
compare_shift_magnitudes(
  x = NULL,
  tree = NULL,
  state_values = NULL,
  measure = c("rate_delta", "percentage_change", "log_ratio"),
  transform = c("log_absolute", "absolute", "signed"),
  tests = "ks",
  alternative = c("two.sided", "less", "greater"),
  ks_simulate_p_value = TRUE,
  ks_B = 10000L,
  bootstrap_p_value = FALSE,
  bootstrap_R = 100L,
  ks_reps = NULL,
  bootstrap_reps = NULL,
  seed = NULL
)
```

Arguments

`x`	A `bifrost_search` object, compatible list, SIMMAP tree, data frame with `rate_change` and the selected `measure`, a list of transition tables or BMM `bifrost_search` objects to pool before testing, a `shift_magnitude_groups` object to compare each named group separately, or `NULL` when using the `tree` argument. Fitted `bifrost_search` inputs are accepted only for multi-regime BMM fits.
`tree`	Optional SIMMAP-style `phylo` tree for generic input mode.
`state_values`	Named numeric state-value vector for generic input mode.
`measure`	Which transition column to compare. `rate_delta` compares fitted child-minus-parent rate differences, `percentage_change` compares percent changes relative to the parent rate, and `log_ratio` compares log child/parent rate ratios.

transform	How to transform the selected measure before testing. "absolute" compares magnitudes, "log_absolute" compares log magnitudes, and "signed" compares signed values.
tests	Character vector of tests to run. Supported values are "t" for a Welch t-test, "wilcox" for an unpaired Wilcoxon rank-sum test, and "ks" for a two-sample Kolmogorov-Smirnov test. Use "all" for all three. The default reproduces the Berv et al. (2026) rate-delta magnitude comparison: a KS test on log(abs(rate_delta)).
alternative	Alternative hypothesis passed to the tests.
ks_simulate_p_value	Logical; passed to stats::ks.test() when "ks" is requested. The default matches the Berv et al. (2026) utility.
ks_B	Integer number of replicates for the simulated KS p-value.
bootstrap_p_value	Logical; estimate the mean p-value from stratified bootstrap resampling within the increase and decrease groups. This reproduces the bootstrap p-value annotation used in the Berv et al. (2026) magnitude-density panels when "ks" is requested.
bootstrap_R	Integer number of bootstrap replicates when bootstrap_p_value = TRUE.
ks_reps	Optional clearer alias for ks_B.
bootstrap_reps	Optional clearer alias for bootstrap_R.
seed	Optional random seed for stochastic KS simulation and bootstrap p-value resampling. The Berv et al. (2026) defaults are unchanged; when a seed is supplied, the previous global RNG state is restored before returning.

Details

Plain lists of transition tables or BMM search objects are pooled into one comparison before testing. Use shift_magnitude_groups() when list elements are analysis groups that should be compared separately. If a group is already a shift_magnitude_comparison, its measure, transform, tests, alternative, KS settings, and bootstrap settings must exactly match the grouped call; otherwise, recompute that group from raw transitions or call compare_shift_magnitudes() with matching settings.

Value

A list of class shift_magnitude_comparison with the transformed values used for testing, group summaries, modal summaries, test results, and settings. When x is a shift_magnitude_groups object, a named shift_magnitude_comparison_set is returned instead.

Examples

```
transitions <- data.frame(
  rate_change = c("increase", "increase", "decrease", "decrease"),
  rate_delta = c(2, 3, -1, -1.5)
)
compare_shift_magnitudes(transitions, ks_reps = 99)
groups <- shift_magnitude_groups(A = transitions, B = transitions)
compare_shift_magnitudes(groups, ks_reps = 99)
```

createSimulationTemplate *Create an Empirically Calibrated Simulation Template for* bifrost

Description

Fit a global single-regime multivariate GLS model to an empirical dataset and convert the fitted object into a reusable calibration template for null and shift-recovery simulation studies.

Usage

```
createSimulationTemplate(
  baseline_tree,
  trait_data,
  formula = "trait_data ~ 1",
  response_columns = NULL,
  predictor_columns = NULL,
  ...
)
```

Arguments

baseline_tree	A rooted phylogenetic tree of class phylo (or coercible via ape::as.phylo()) whose unique, non-empty tip labels match the row names of trait_data.
trait_data	A numeric matrix or data frame containing the empirical data used to parameterize the simulations. Row names must be unique, non-empty, and match baseline_tree$tip.label. Column names may be omitted for matrix inputs; when present, they must be unique and non-empty.
formula	Formula specification passed to mvMORPH::mvgls(). May be a single character string or a formula object. Indexed formulas that reference trait_data directly are supported (for example, "trait_data ~ 1" or "trait_data[, 1:12] ~ trait_data[, 13]"), as are named-column formulas such as cbind(y1, y2) ~ size + grp.
response_columns	Optional column specification identifying the multivariate response. May be integer positions, character column names, or a mix resolvable against trait_data. Because R coerces mixed atomic vectors to character, integer-like strings are treated as positions when they are not exact column names. Numeric positions must be whole numbers. For intercept-only workflows this defaults to all columns. For any workflow where the response is a subset of trait_data, including formulas such as "trait_data[, 1:12] ~ 1", this should be supplied explicitly.
predictor_columns	Optional column specification identifying predictor columns used by the global calibration model. For named-column formulas these are usually inferred from the formula itself. For non-intercept workflows where predictor_columns is

omitted, the complement of `response_columns` is used unless the formula identifies raw predictor columns directly.

`...` Additional arguments passed to `mvMORPH::mvgls()` for the global empirical fit (for example `method = "LL"` or `error = TRUE`).

Details

The returned template stores:

- the aligned empirical tree and calibration data,
- the fitted global `mvgls` model,
- evaluated copies of key fit settings (`fit_method`, `fit_error`) that can be reused safely by downstream simulation-study wrappers,
- the fitted response mean structure,
- the full empirical residual covariance and residual degrees of freedom used by the empirical Wishart generator, and
- summaries of the diagonal and off-diagonal elements (`variance_mean`, `variance_sd`, `covariance_mean`, `covariance_sd`) retained by the manuscript-reproduction generator.

The template is a calibration object, not a full simulation-study design. Its formula defines the one global mean model used to estimate fitted values and residual covariance structure. Downstream simulation studies then regenerate the response block around those fitted means and evaluate intercept-only shift-search behavior on the simulated responses. In other words, a richer calibration model can still feed the response-only `trait_data ~ 1` simulation workflow. Predictor columns are retained in the template for calibration provenance, but simulated replicates returned by the study helpers contain the regenerated response block only.

Internally, all supported formulas are normalized into a single simulation formula specification. Indexed `trait_data` formulas are rewritten onto named columns, formula objects are accepted alongside character strings, predictor schemas record numeric/logical/factor/ordered predictors, and unsupported forms such as transformed responses, `.` shorthand, and character predictors are rejected early.

Value

A list of class `bifrost_simulation_template` containing the aligned inputs, fitted global model, empirical mean structure, response/predictor column metadata, stored fit settings, the empirical residual covariance and degrees of freedom, and covariance summaries used for simulation.

See Also

`simulateNullDataset()`, `simulateShiftedDataset()`, `runFalsePositiveSimulationStudy()`, `runShiftRecoverySimulationStudy()`, `searchOptimalConfiguration()`

Examples

```
set.seed(1)
tr <- ape::rtree(12)
X <- matrix(rnorm(12 * 2), ncol = 2)
```

```
rownames(X) <- tr$tip.label

tmpl <- createSimulationTemplate(
  baseline_tree = tr,
  trait_data = X,
  formula = "trait_data ~ 1",
  method = "LL"
)

tmpl
```

---

distribution-fit-data-frames
Coerce Shift And Distribution Summaries To Data Frames

Description

Extract concise, vignette-friendly tables from fitted shift-magnitude and distribution objects. These methods do not change the underlying objects; they only select and lightly round commonly inspected columns.

Usage

```
## S3 method for class 'shift_magnitude_comparison'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("summary", "tests", "modal", "ks"),
  analysis = NULL,
  digits = 4L,
  ...
)

## S3 method for class 'shift_magnitude_comparison_set'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("ks", "summary", "tests", "modal"),
  digits = 4L,
  ...
)

## S3 method for class 'shift_magnitude_counts'
as.data.frame(x, row.names = NULL, optional = FALSE, digits = 4L, ...)
```

```
## S3 method for class 'shift_magnitude_count_set'
as.data.frame(x, row.names = NULL, optional = FALSE, digits = 4L, ...)

## S3 method for class 'rate_distribution_fit'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("rankings", "parameters", "goodness"),
  models = NULL,
  n = Inf,
  digits = 4L,
  ...
)

## S3 method for class 'waiting_time_distribution_fit'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("rankings", "parameters", "lineage", "lineage_parameters"),
  models = NULL,
  n = Inf,
  digits = 4L,
  ...
)
```

Arguments

x	A shift_magnitude_comparison, shift_magnitude_comparison_set, shift_magnitude_counts, shift_magnitude_count_set, rate_distribution_fit, or waiting_time_distribution_fit object.
row.names, optional	Included for compatibility with base::as.data.frame(); ignored.
component	Summary table to extract. Ignored for shift-magnitude count objects. For shift-magnitude comparisons and comparison sets, use "summary", "tests", "modal", or "ks". For distribution fits, use "rankings" or "parameters"; rate-distribution fits with bootstrapped Gumbel diagnostics also support "goodness", which currently reports the Gumbel Kullback-Leibler divergence summary. For waiting-time fits with by_lineage = TRUE, use "lineage" or "lineage_parameters".
analysis	Optional label used when component = "ks" for shift_magnitude_comparison objects.
digits	Number of significant digits used for numeric columns.
...	Reserved for future extensions. Supplying unused arguments is an error.
models	Optional model identifiers used to filter parameter tables.
n	Maximum number of rows to return.

Value

A data.frame whose columns depend on component and the input class. Shift-magnitude summary tables report directional sample sizes and moments; test tables report the requested test statistics and p-values; modal tables report KDE modes and ratios; and component = "ks" returns one KS-focused row per comparison. Distribution-fit ranking tables report model information criteria, parameter tables report model parameters and estimates, Gumbel goodness tables report bootstrapped KL-divergence summaries, and waiting-time lineage tables report per-lineage fit summaries.

`evaluateShiftRecovery` *Evaluate Shift Recovery Against Simulated Ground Truth*

Description

Compare inferred shift locations against known simulated shift locations across multiple datasets using both strict node matching and fuzzy matching within a configurable node distance. This is a package-clean export of the manuscript's evaluate_shift_recovery() function.

Usage

```
evaluateShiftRecovery(
  simdata,
  simresults,
  fuzzy_distance = 2,
  weighted = TRUE,
  verbose = TRUE
)
```

Arguments

simdata	A list of simulated shifted datasets, typically the simdata component returned by runShiftRecoverySimulationStudy(). Each element must contain at least paintedTree and shiftNodes.
simresults	A list of search results corresponding to simdata, usually the results component returned by runShiftRecoverySimulationStudy(). Each element should contain shift_nodes_no_uncertainty and num_candidates. Results from searchOptimalConfiguration() also carry candidate_nodes, which allows candidate-aware specificity calculations; legacy result objects without that component retain the historical count-only calculation. If weighted = TRUE, ic_weights is also used when available.
fuzzy_distance	Integer node distance threshold used for fuzzy matching.
weighted	Logical; if TRUE, compute weighted precision/recall/F1 using the inferred-node IC weights from each search result when available.
verbose	Logical; if TRUE, print a compact summary of the strict, fuzzy, and weighted metrics.

Details

The evaluation uses three complementary summaries:

- strict metrics count only exact node matches,
- fuzzy metrics allow inferred shifts within `fuzzy_distance` nodes of a true shift, using a greedy one-to-one assignment, and
- weighted metrics apply IC weights only to inferred nodes, following the simulation-study implementation.

Search results carrying a non-empty `error` field are retained by the study wrappers for diagnosis but excluded from recovery counts and metrics. In otherwise complete, successful results, an explicitly present `shift_nodes_no_uncertainty = NULL` means no inferred shifts and contributes false negatives for the true shifts. A missing field remains incomplete and is excluded. Empty integer vectors are handled identically to explicit `NULL`.

For current search results, true-negative counts are calculated over the explicit `candidate_nodes` universe. True shifts excluded by the search's candidate filter still contribute false negatives, but are not subtracted from the number of candidate negatives. This prevents specificity and balanced accuracy from being distorted when a simulated shift is not an eligible search candidate. Supplied candidate-node vectors must contain unique, valid whole-number node IDs and agree with `num_candidates`. For compatibility, result objects that provide only `num_candidates` use the historical count-only formula.

F1 is calculated directly from counts as `2 * TP / (2 * TP + FP + FN)`. It is zero when no true shifts are recovered but false positives or false negatives exist, including successful searches with no inferred shifts. It remains `NA` when both true and inferred shifts are absent, or when no records can be evaluated. Precision and recall retain their own undefined cases. Weighted F1 uses `2 * weighted_TP / (weighted_TP + weighted_FP + n_true_shifts)` and is `NA` if inferred-node IC weights are unavailable.

When a replicate has no evaluable candidate shifts (`num_candidates == 0`), recall-style quantities can still be computed from the true and inferred shifts, but specificity, false-positive rate, and balanced accuracy are returned as `NA` because there are no evaluable negatives.

Value

A list of class `bifrost_shift_recovery_evaluation` with five components:

`strict` Strict precision, recall, F1, specificity, false-positive rate, and balanced accuracy.

`fuzzy` The same metrics under fuzzy matching.

`weighted` Weighted strict and fuzzy precision/recall/F1 summaries when `weighted = TRUE`; otherwise `NULL`.

`counts` Aggregated contingency-table counts for the strict and fuzzy matching schemes.

`n_evaluable_replicates` Number of replicate pairs included in the aggregated counts after failed or structurally incomplete records are excluded.

See Also

`runShiftRecoverySimulationStudy()`, `simulateShiftedDataset()`

Examples

```
set.seed(1)
tr <- ape::rtree(8)
tr <- phytools::paintSubTree(
  tr,
  node = ape::Ntip(tr) + 1L,
  state = "ancestral",
  anc.state = "ancestral"
)
true_node <- setdiff(unique(tr$edge[, 1]), ape::Ntip(tr) + 1L)[1]
simdata <- list(list(paintedTree = tr, shiftNodes = true_node))
simresults <- list(list(
  shift_nodes_no_uncertainty = true_node,
  num_candidates = 6L,
  ic_weights = data.frame(
    node = true_node,
    ic_weight_withshift = 0.8
  )
))

recovery <- evaluateShiftRecovery(
  simdata,
  simresults,
  fuzzy_distance = 2,
  verbose = FALSE
)
recovery$strict
```

---

fisher_z_transform *Fisher's Z Transformation*

Description

Transform correlation coefficients with `atanh(r)` while rejecting values outside the open interval `(-1, 1)`.

Usage

```
fisher_z_transform(correlations)
```

Arguments

`correlations`	Numeric correlation vector.

Value

Numeric vector of transformed correlations.

Examples

```
fisher_z_transform(c(-0.5, 0, 0.5))
```

`fit_rate_distribution` *Fit Candidate Distributions To Rates*

Description

Fit and rank candidate continuous distributions for a numeric vector of rates, a lineage-rate table, or the fitted regime rates in a `bifrost_search` object. `bifrost_search` inputs must be multi-regime BMM fits; other model families are rejected before fitting. The function is generic with respect to the candidate distribution: no distribution, including Gumbel, receives special handling unless requested through `models`.

Usage

```
fit_rate_distribution(
  x,
  log = TRUE,
  models = NULL,
  select_by = c("AIC", "BIC")
)
```

Arguments

`x`	Numeric vector, lineage-rate table, or `bifrost_search` object.
`log`	Logical; log-transform positive rate values before fitting. Lineage-rate tables with a `log_lineage_rate` column are treated as already log transformed when `log = TRUE`. When `log = FALSE`, lineage-rate tables must include a raw rate column rather than only `log_lineage_rate`.
`models`	Character vector of `univariateML` model identifiers. When `NULL`, a broad real-valued candidate set is used.
`select_by`	Criterion used to select `selected_model`.

Value

A list of class `rate_distribution_fit` with the transformed data, model rankings, a long-form `parameters` table of fitted parameter estimates, individual fitted objects, selected model, and failed model attempts. The raw `univariateML` objects remain available in `fits` for workflows that need package-specific methods.

Examples

```
if (requireNamespace("univariateML", quietly = TRUE)) {
  rates <- c(0.8, 1.1, 1.4, 1.9, 2.5, 3.1)
  fit_rate_distribution(rates, models = c("norm", "gumbel"))
}
```

`fit_regime_covariances` *Fit Independent Post-hoc Regime Covariance Models*

Description

Fit separate `mvMORPH::mvgls()` Brownian-motion models to the tips assigned to each contemporary mapped regime. This is a post-hoc workflow: it estimates independent regime covariance matrices after a mapped-regime analysis has already been chosen.

Usage

```
fit_regime_covariances(
  x = NULL,
  tree = NULL,
  trait_data,
  formula = trait_data ~ 1,
  model = "BM",
  min_tips = 2L,
  cores = 1L,
  error = TRUE,
  ...
)
```

Arguments

`x`	Optional `bifrost_search` object, compatible list with a `tree_no_uncertainty_untransformed` component, or SIMMAP-style `phylo` tree. If `tree` is supplied, `x` is only used as an optional source of regime-rate parameters.
`tree`	Optional SIMMAP-style `phylo` tree with mapped states, non-empty regime-state identifiers, and unique, non-empty tip labels. Required when `x` is `NULL`.
`trait_data`	Matrix or data frame with unique, non-empty row names matching the tree tip labels. Column names may be omitted; when present, they must also be unique and non-empty.
`formula`	Formula used for each regime-specific `mvgls()` fit. The default treats `trait_data` as the multivariate response. Formulae may refer to `trait_data` directly, as in `trait_data[, 1:5] ~ trait_data[, 6]`, or to columns by name.
`model`	Evolutionary model passed to `mvMORPH::mvgls()`. The default `"BM"` matches the independent post-hoc covariance workflow.
`min_tips`	Minimum number of tips required before a regime is fitted. Regimes are fitted when `tip_count >= min_tips`; regimes with fewer tips are returned with status `"skipped"`. The default of two tips matches the Berv et al. post-hoc fitting script.
`cores`	Number of workers. Values greater than one use `future.apply::future_lapply()` with a temporary multisession plan.

error	Logical passed to mvMORPH::mvgls().
...	Additional arguments passed to mvMORPH::mvgls().

Details

The covariance matrices returned here are independent post-hoc estimates. They are not the same object as search$VCVs in a bifrost_search result: the current bifrost search model uses scalar transformations of a shared phenotypic covariance matrix, so search$VCVs are proportional joint-model matrices rather than independently refit regime matrices. Extracted matrices must be symmetric and positive semidefinite, contain finite entries and strictly positive diagonal variances, and, when named, have unique matching row and column trait names. Fits that do not satisfy those requirements are retained with status "failed" and a diagnostic message.

Value

An object of class regime_covariances, a list with:

fits Named list of regime-specific mvgls fits or NULL.

covariances Named list of extracted post-hoc covariance matrices or NULL for skipped/failed regimes.

status Data frame with regime ID, tip count, regime age, status, and message.

rates Named regime-rate vector if available from x; otherwise NULL.

Examples

```
tree <- ape::read.tree(text = paste0(
  "(((a:1,b:1):1,(c:1,d:1):1):1,",
  "((e:1,f:1):1,(g:1,h:1):1):1);"
))
tree <- phytools::paintSubTree(
  tree, node = ape::Ntip(tree) + 1L, state = "root"
)
tree <- phytools::paintSubTree(
  tree, node = ape::getMRCA(tree, c("a", "d")), state = "slow"
)
tree <- phytools::paintSubTree(
  tree, node = ape::getMRCA(tree, c("e", "h")), state = "fast"
)
set.seed(1)
traits <- matrix(
  stats::rnorm(16),
  nrow = 8,
  dimnames = list(tree$tip.label, c("bill", "wing"))
)
fits <- fit_regime_covariances(
  tree = tree,
  trait_data = traits,
  min_tips = 4,
  error = FALSE,
  method = "LL"
)
fits$status
```

fit_regime_covariance_runs *Fit Post-hoc Regime Covariance Models Across Runs*

Description

Apply `fit_regime_covariances()` to each search/run in a named list. This is the package analogue of the manuscript `fitPosthocModels()` step: the top-level run iteration is explicit, while each run can still fit its regime subtrees in parallel through `cores`.

Usage

```
fit_regime_covariance_runs(
  x,
  trait_data,
  formula = trait_data ~ 1,
  model = "BM",
  min_tips = 2L,
  cores = 1L,
  error = TRUE,
  tree_element = "tree_no_uncertainty_untransformed",
  verbose = FALSE,
  ...
)
```

Arguments

`x`	Named list of `bifrost_search` objects, compatible search-like lists, or SIMMAP-style `phylo` trees. Missing or blank names are generated and names must be unique after normalization.
`trait_data`	Matrix or data frame with unique, non-empty row names matching the tree tip labels. Column names may be omitted; when present, they must also be unique and non-empty.
`formula`	Formula used for each regime-specific `mvgls()` fit. The default treats `trait_data` as the multivariate response. Formulae may refer to `trait_data` directly, as in `trait_data[, 1:5] ~ trait_data[, 6]`, or to columns by name.
`model`	Evolutionary model passed to `mvMORPH::mvgls()`. The default `"BM"` matches the independent post-hoc covariance workflow.
`min_tips`	Minimum number of tips required before a regime is fitted. Regimes are fitted when `tip_count >= min_tips`; regimes with fewer tips are returned with status `"skipped"`. The default of two tips matches the Berv et al. post-hoc fitting script.
`cores`	Number of workers. Values greater than one use `future.apply::future_lapply()` with a temporary multisession plan.

error	Logical passed to mvMORPH::mvgls().
tree_element	For search-like list inputs, the element containing the mapped tree to use for post-hoc refits. Set to NULL to let fit_regime_covariances() resolve the tree from each run directly.
verbose	Logical; if TRUE, print progress messages by run name.
...	Additional arguments passed to mvMORPH::mvgls().

Value

A named list of regime_covariances objects with class regime_covariance_runs.

Examples

```
tree <- ape::read.tree(text = paste0(
  "(((a:1,b:1):1,(c:1,d:1):1):1,",
  "((e:1,f:1):1,(g:1,h:1):1):1);"
))
tree <- phytools::paintSubTree(
  tree, node = ape::Ntip(tree) + 1L, state = "root"
)
tree <- phytools::paintSubTree(
  tree, node = ape::getMRCA(tree, c("a", "d")), state = "slow"
)
tree <- phytools::paintSubTree(
  tree, node = ape::getMRCA(tree, c("e", "h")), state = "fast"
)
set.seed(2)
traits <- matrix(
  stats::rnorm(16),
  nrow = 8,
  dimnames = list(tree$tip.label, c("bill", "wing"))
)
run_fits <- fit_regime_covariance_runs(
  list(first = tree, second = tree),
  trait_data = traits,
  min_tips = 4,
  error = FALSE,
  method = "LL"
)
vapply(run_fits, function(x) sum(x$status$status == "ok"), integer(1))
```

---

fit_waiting_time_distribution

Fit Candidate Distributions To Shift Waiting Times

Description

Fit and rank candidate positive continuous distributions for shift waiting times. Inputs can be numeric waiting times, data frames with a `TimeToNext` column, BMM `bifrost_search` output, or the output of `shift_waiting_times()`. Fitted `bifrost_search` inputs are accepted only for multi-regime BMM fits; numeric vectors and data frames remain generic. Because the candidate distributions have strictly positive support, inputs containing zero waits from tied or simultaneous events are rejected with an explanatory error; the zero waits remain available from `shift_waiting_times()` for descriptive analysis.

Usage

```
fit_waiting_time_distribution(
  waiting_times,
  by_lineage = FALSE,
  global_include_root_wait = TRUE,
  models = c("exp", "gamma", "weibull", "lnorm", "invgauss"),
  select_by = c("AIC", "BIC")
)
```

Arguments

`waiting_times`	Numeric vector, waiting-time data frame, `shift_waiting_times` object, or BMM `bifrost_search` output.
`by_lineage`	Logical; when `TRUE`, use pooled between-shift lineage intervals for the top-level fit instead of chronological whole-tree intervals, and also fit candidate distributions separately for each lineage with at least two waiting times.
`global_include_root_wait`	Logical; when fitting whole-tree global waits from a `shift_waiting_times` object, `bifrost_search` object, or global waiting-time data frame, include the initial root-to-first-shift interval. The default `TRUE` matches the Berv et al. (2026) reported whole-tree Weibull summary. Set to `FALSE` to exclude the row labeled `PreviousShift == "root"`. Multiple such rows are ambiguous and produce an error; data without that label are unchanged.
`models`	Character vector of positive-support `univariateML` model identifiers.
`select_by`	Criterion used to select best models.

Value

A list of class `waiting_time_distribution_fit` with pooled model rankings and a long-form `parameters` table of fitted parameter estimates. With `by_lineage = FALSE`, the top-level pool contains chronological whole-tree intervals; with `by_lineage = TRUE`, it contains pooled between-shift lineage intervals and the result additionally includes per-lineage best-model, parameter, and exponential-rate summaries. The raw `univariateML` objects remain available in `fits` and `lineage_fits`.

Examples

```
if (requireNamespace("univariateML", quietly = TRUE)) {
  waits <- c(0.5, 0.8, 1.2, 1.9, 2.4, 3.0)
```

```
  fit_waiting_time_distribution(waits, models = c("exp", "gamma", "weibull"))
}
```

`generateViridisColorScale` *Generate Viridis Color Palette for Ranked Parameters*

Description

Creates a named color mapping for a set of numeric parameters (e.g., evolutionary rates) using the **viridis** color palette. Parameters are sorted in ascending order and assigned evenly spaced colors by rank. Numeric magnitude and distance between parameter values are not encoded in the colors.

Usage

```
generateViridisColorScale(params)
```

Arguments

`params`	A named numeric vector of parameter values (e.g., rates). The names will be preserved and used to label the resulting color mapping.

Details

This function is useful for plotting results where parameters should be visually distinguished by their ordering (e.g., rate shifts across a phylogeny). By using the perceptually uniform viridis palette, it avoids misleading color interpretations common with rainbow scales. Colors encode sorted rank only; they do not represent the magnitude of a parameter or the distance between parameter values.

Value

A named list with two elements:

`NamedColors` A named character vector of hex color codes, with names corresponding to the input parameter names, ordered by increasing parameter value and colored at evenly spaced positions by rank.

`ParamColorMapping` A named numeric vector of the sorted parameter values, maintaining the same order and names as `NamedColors`.

See Also

`viridis::viridis()` for details on the color palette.

Examples

```
if (requireNamespace("viridis", quietly = TRUE)) {
  library(viridis)
  set.seed(1)
  rates <- c(A = 0.1, B = 0.5, C = 0.9)
  color_scale <- generateViridisColorScale(rates)

  # View the color assignments
  color_scale$NamedColors

  # Plot with colors
  barplot(color_scale$ParamColorMapping,
          col = color_scale$NamedColors,
          main = "Rates with Viridis Colors")
}
```

---

icTrajectory *Create an Information-Criterion Trajectory*

Description

Extract a standardized information-criterion (IC) trajectory from a completed `bifrost_search` object or a compatible search-result list.

Usage

```
icTrajectory(x, baseline_ic = NULL, ...)
```

Arguments

`x`	A `bifrost_search` object returned by `searchOptimalConfiguration()` with `store_model_fit_history = TRUE`, or a compatible list with `baseline_ic` and `model_fit_history`.
`baseline_ic`	Optional finite numeric baseline IC. When supplied, this value is used for `step = 0` instead of `x$baseline_ic`. This is useful for legacy search-like lists that did not store `baseline_ic`.
`...`	Reserved for future extensions.

Details

`icTrajectory()` is a lightweight extractor. It does not refit models or recompute the search. The baseline IC is taken from `x$baseline_ic` unless `baseline_ic` is supplied, and is always included as `step = 0`; later rows summarize the stored proposal history in `x$model_fit_history$fits`. When a legacy object stores a historical `x$model_fit_history$ic_acceptance_matrix`, those matrix values are used to fill missing proposal IC or acceptance values. Objects that only contain the legacy matrix are also supported. In legacy cases without explicit proposal metadata,

candidate_node cannot be recovered, and regime_id is inferred from proposal order because bifrost assigns candidate shift regimes sequentially during the search.

The returned object is a data frame with class c("icTrajectory", "data.frame") and the following columns:

- step: search step, with 0 reserved for the baseline.
- ic: IC for the baseline or evaluated proposal.
- accepted: proposal decision; NA for the baseline.
- best_ic: running best IC after each step.
- delta_ic: proposal improvement over the current best before the proposal; positive values indicate improvement.
- status: "baseline", "accepted", "rejected", or "error".
- candidate_node: phylogenetic node where the shift was evaluated.
- regime_id: mapped-state/regime label assigned to the evaluated candidate shift. The baseline row uses "0".
- error: optional error message.

Value

A data frame with class c("icTrajectory", "data.frame").

Examples

```
search <- list(
  baseline_ic = -1000,
  IC_used = "GIC",
  model_fit_history = list(
    fits = list(
      list(
        step = 1, candidate_node = 42, regime_id = "1",
        ic = -1010, accepted = TRUE, delta_ic = 10
      ),
      list(
        step = 2, candidate_node = 57, regime_id = "2",
        ic = -1008, accepted = FALSE, delta_ic = -2
      )
    )
  )
)
class(search) <- c("bifrost_search", "list")
traj <- icTrajectory(search)
legacy_traj <- icTrajectory(search, baseline_ic = -995)
plot(traj)
```

lineage_rates *Compute Weighted Lineage Rates*

Description

Compute the lineage-rate summary statistic described by Berv et al. (2026) from mapped-state phylo/simmap tree objects (Paradis and Schliep 2019; Revell 2012, 2024), including the mapped trees returned by bifrost. The function calculates present-biased weighted summaries of the state values inherited at ancestral nodes along each root-to-tip path. For bifrost BMM fits, the state values are the scalar regime-rate parameters stored in bifrost_search$model_no_uncertainty$param; for generic input, set bifrost_search = NULL and provide tree with a named state_values vector. Results include shift counts, terminal-state values, and the summary statistic, returned on the log scale by default.

Usage

```
lineage_rates(
  bifrost_search = NULL,
  tree = NULL,
  state_values = NULL,
  decay_base = 2,
  half_life = 5,
  normalize_weights = TRUE,
  age_reference = c("tree", "tip"),
  log = TRUE,
  cores = 1L,
  progress = interactive()
)
```

Arguments

bifrost_search	A bifrost_search object returned by bifrost, a plain list with the required bifrost components, or NULL. bifrost inputs must include a fitted model_no_uncertainty component that identifies a multi-regime Brownian-motion model (model = "BMM") and provides named numeric regime-rate parameters in bifrost_search$model_no_uncertainty$para Use NULL for generic input mode, where tree and state_values are required. Supply either bifrost_search or both tree and state_values, but not both input modes at once. Plain-list bifrost inputs must include tree_no_uncertainty_untransformed and model_no_uncertainty.
tree	SIMMAP-style phylo/simmap tree (Paradis and Schliep 2019; Revell 2012, 2024) required when bifrost_search = NULL for generic input mode. Must be NULL when bifrost_search is supplied. The tree must include SIMMAP maps state data from which node and tip states can be recovered. For the age-decay interpretation, the tree should be time-scaled; non-ultrametric trees are accepted, with age calculations controlled by age_reference. Internal nodes must use the conventional phylo numbering in which the root is Ntip(tree) + 1.

`state_values` Named numeric vector required in generic input mode and invalid for `bifrost` inputs. Names must include every SIMMAP state label in the tree; extra names are ignored. Values must be finite, and must be strictly positive when `log = TRUE`.

`decay_base` Numeric exponential-decay base `b` in the weighting kernel. Must be at least `1`. Larger values make weights decline more strongly as elements get farther from the present; `decay_base = 1` gives equal age weights regardless of `half_life`.

`half_life` Optional decay time scale `T`, in the same time units as the tree. With the default `decay_base = 2`, `T` is a true half-life: an element `T` time units farther from the present receives half the unnormalized weight. More generally, weights are multiplied by `1 / decay_base` over each interval of length `T`. Set `half_life = NULL` to use the unscaled kernel `b^(-age)`; this only changes the weights when `decay_base > 1`.

`normalize_weights` Logical; divide raw age-decay weights by their lineage-specific sum before averaging. Berv et al. (2026) used `TRUE`, which keeps the statistic as a weighted mean and makes lineages comparable when root-to-tip paths include different numbers of ancestral nodes. If every included ancestral node has the same state, the normalized value equals that state's supplied value or fitted rate.

`age_reference` Character; reference point for converting node heights into ages before the present. The default, `"tree"`, measures ages relative to the maximum tree height and preserves the original global-present behavior. Use `"tip"` for heterochronous or otherwise non-ultrametric trees when recency should be measured before each terminal tip's own sampling point. These choices are identical for ultrametric trees and mainly affect results when `normalize_weights = FALSE`.

`log` Logical; return log-scale lineage-rate columns by default. `bifrost` BMM rates must always be finite and strictly positive. Generic `state_values` must be strictly positive when `log = TRUE`; when `log = FALSE`, generic values can remain on their original scale.

`cores` Integer number of future workers. Values greater than one use `future::multisession` through `future.apply::future_lapply()`, which is platform agnostic.

`progress` Logical; show a local `progressr` text progress bar for the tip-wise computations. Defaults to `interactive()`, so progress is shown in interactive sessions and suppressed in non-interactive runs.

Details

Overview: The summary statistic follows Berv et al. (2026) as a descriptive, present-biased summary of each tip's inferred mapped-state history. In the default BMM use case, it summarizes heterogeneous phenotypic tempo along a tip-to-root path while giving more influence to recently inherited regimes. The calculation uses the node and tip states recovered from the supplied SIMMAP-style mapped-state tree. The parameters `decay_base`, `half_life`, and `normalize_weights` define the weighting kernel used to explore how inferred lineage summaries change as deeper history is weighted more or less strongly.

In the default `bifrost` mode, `lineage_rates()` reads regime-specific Brownian variance rates from `bifrost_search$model_no_uncertainty$param` and summarizes how those rates are inherited along mapped root-to-tip histories. That interpretation requires a heterogeneous Brownian-motion model: a multi-regime BMM fit with at least two mapped states.

Method and formula: Using the notation of Berv et al. (2026), generalized to an arbitrary decay base b and state-associated value r_i, the normalized summary statistic is

$$\bar{r}_{\mathrm{WLR}} = \frac{1}{Z} \sum_{i=1}^{L} b^{-a_i/T} \cdot r_i$$

where the normalizing constant is

$$Z = \sum_{j=1}^{L} b^{-a_j/T}.$$

Here, $\bar{r}_{\mathrm{WLR}}$ is the weighted mean lineage value, r_i is the state-associated value for node or path element i, a_i is that element's age before the selected reference point, b is `decay_base`, T is the decay time scale, and L is the number of nodes or path elements included in the statistic. For the branch-based summary statistic aligned with Berv et al. (2026), i indexes ancestral nodes along the lineage, including the root and excluding the terminal tip state. This is the expected structure of `bifrost` mapped trees, where regime changes are represented by changes in node states rather than integrated over within-branch SIMMAP segments.

In `bifrost` output, r_i is represented by the scalar BMM regime-rate parameters in `bifrost_search$model_no_uncertain` following Berv et al. (2026), these correspond to the arithmetic mean of the diagonal variance terms in each regime's estimated evolutionary variance-covariance matrix. This is the weighted phenotypic-rate interpretation used in Berv et al. (2026). In generic input mode, r_i is the corresponding value from `state_values`.

Weighting, normalization, and age reference: Berv et al. (2026) used the base-2 case, and the defaults (`decay_base = 2`, `half_life = 5`, and `normalize_weights = TRUE`) preserve that weighting scheme. With these defaults, a regime that is `half_life` time units farther from the present receives half the unnormalized weight. `decay_base = 1` gives every element raw weight 1 regardless of `half_life`, so the summary statistic becomes an equal-weighted mean when `normalize_weights = TRUE`. Values below 1 are rejected because they would give greater weight to regimes farther from the present.

The kernel terms $b^{-a_i/T}$ are raw age-decay weights. When `normalize_weights = TRUE`, the function divides those raw weights by Z within each lineage before averaging. This keeps the statistic as a weighted mean of state-associated values, rather than letting lineages with more ancestral nodes accumulate larger values simply because more terms were summed. If every included ancestral node is assigned to the same state, the normalized value equals that state's supplied value or fitted rate regardless of how many nodes occur along the path. When `normalize_weights = FALSE`, the $1/Z$ normalization is skipped.

The mapped tree should be time-scaled for the age-decay weights to have a direct temporal interpretation. The `age_reference` argument controls how the ages a_i are measured. With the default `age_reference = "tree"`, ages are measured relative to the maximum tree height, matching the original global-present interpretation. For heterochronous or otherwise non-ultrametric trees, `age_reference = "tip"` measures recency before each terminal tip's own sampling point. These choices are identical for ultrametric trees. When `normalize_weights = TRUE`, they give the same top-level normalized values because all raw weights within a lineage are rescaled by the same constant before normalization. The choice mainly affects results when `normalize_weights = FALSE`.

When half_life = NULL, the unscaled kernel b^{-a_i} is used instead of $b^{-a_i/T}$. This removes the half-life divisor from the exponent and only changes the weights when decay_base > 1. Parameter combinations that make a mathematically positive decay weight unrepresentable in double precision are rejected rather than returning a non-finite summary.

Interpretation: Interpret this as a flexible descriptive statistic for an inferred mapped history, not as a model-derived estimator with a unique optimal weighting scheme. It is not an instantaneous tip-rate estimate or a strict recent-time-window average. The default statistic is node-based: it uses ancestral node states along each root-to-tip path and does not integrate over within-edge SIMMAP segment durations. For SIMMAP-style trees with mapped changes inside branches, those segment durations are not part of the top-level statistic, and within-edge changes are not counted as shifts unless they are represented by parent-child node-state differences.

Lineage and shift diagnostics: The top-level log_lineage_rate or lineage_rate column is computed from ancestral node states along each root-to-tip path. Because the statistic is node-based and normalized within each lineage, its temporal resolution depends on the ages and density of ancestral nodes. Lineages with many recent nodes can downweight regimes farther from the present more rapidly, whereas sparse lineages may retain more influence from regimes farther from the present.

The branch_metrics and shift_metrics attributes store concise unweighted shift diagnostics: the number of detected parent-child node-state changes, root-to-tip time, shifts per unit time, and shifts per speciation event. branch_metrics additionally stores the weighted ancestral-node intermediate used for the top-level lineage-rate column. tip_rate is the fitted rate or supplied value for the terminal SIMMAP state. Compare the terminal value to the lineage-rate columns when you want to assess how much the inherited state history differs from the terminal regime state.

Generic input mode: lineage_rates() can also operate directly on a SIMMAP-style tree whose mapped node states refer to externally estimated values supplied through state_values. In this generic input mode, no fitted model is inspected: the same node-based summary statistic is applied to the user-supplied state values. This can be useful for summarizing arbitrary state-associated quantities on a mapped tree when the node-state interpretation is appropriate. The output column names retain the rate terminology used by the bifrost workflow for compatibility, but the values represent the supplied state_values and their interpretation depends on what those values measure. For generic SIMMAP trees, use this helper when ancestral node states are the summary target; it is not a duration-weighted integrator over mapped within-edge segments.

For a completed search result named search_result, the usual search-result call is lineage_rates(bifrost_search = search_result).

Value

A data.frame with one row per tip. The stable top-level columns are tip.label, shift_count, the ancestral-node summary statistic, tip_state, and tip_rate. With log = TRUE, the summary statistic is named log_lineage_rate, and log_tip_rate is included. With log = FALSE, the summary statistic is named lineage_rate.

Additional diagnostic tables are stored as attributes. branch_metrics contains the per-tip intermediate table for the top-level ancestral-node statistic. Its columns are Tip, Shift_Count, Total_Time, Shift_Rate_Per_Time, Shift_Rate_Per_Speciation, Weighted_Lineage_Value,

`Tip_State`, and `Tip_State_Value`. `shift_metrics` contains the same concise shift-count and time diagnostics without `Weighted_Lineage_Value`; its columns are `Tip`, `Shift_Count`, `Total_Time`, `Shift_Rate_Per_Time`, `Shift_Rate_Per_Speciation`, `Tip_State`, and `Tip_State_Value`. `Shift_Count` and both shift-rate columns are unweighted counts or ratios of detected parent-child node-state transitions. The `settings` attribute records the resolved input mode, model family, tree name, weighting parameters, age reference, log setting, and execution settings used for the calculation.

References

Berv, J. S. et al. (2026). Rates of passerine body plan evolution in time and space. *Nature Ecology & Evolution*. doi:10.1038/s41559-026-03110-5.

Clavel, J., Escarguel, G., and Merceron, G. (2015). mvMORPH: an R package for fitting multivariate evolutionary models to morphometric data. *Methods in Ecology and Evolution*, 6, 1311-1319. doi:10.1111/2041-210X.12420.

Clavel, J., Aristide, L., and Morlon, H. (2019). A penalized likelihood framework for high-dimensional phylogenetic comparative methods and an application to new-world monkeys brain evolution. *Systematic Biology*, 68, 93-116. doi:10.1093/sysbio/syy045.

Paradis, E., and Schliep, K. (2019). ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35, 526-528. doi:10.1093/bioinformatics/bty633.

Revell, L. J. (2012). phytools: an R package for phylogenetic comparative biology (and other things). *Methods in Ecology and Evolution*, 3, 217-223. doi:10.1111/j.2041-210X.2011.00169.x.

Revell, L. J. (2024). phytools 2.0: an updated R ecosystem for phylogenetic comparative methods (and other things). *PeerJ*, 12, e16505. doi:10.7717/peerj.16505.

Examples

```
# Generic input mode: summarize arbitrary state-associated values on a mapped tree.
toy_tree <- ape::read.tree(text = "((a:1,b:1):1,c:2);")
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
state_values <- c("0" = 1.2, "1" = 3.4)
lineage_rates(
  tree = toy_tree,
  state_values = state_values,
  log = FALSE,
  progress = FALSE
)
```

`plot.icTrajectory` *Plot an Information-Criterion Trajectory*

Description

Plot IC scores and the running best IC across a stored `bifrost` search trajectory.

Usage

```
## S3 method for class 'icTrajectory'
plot(
  x,
  main = "IC Trajectory",
  show_delta = "overlay",
  ic_limits = NULL,
  delta_limits = NULL,
  xlab = "Search step",
  ylab = NULL,
  symbols = NULL,
  scales = NULL,
  point_sizes = NULL,
  line_widths = NULL,
  text_sizes = NULL,
  annotation = NULL,
  legend = TRUE,
  ...
)
```

Arguments

`x`	An `icTrajectory` object.
`main`	Plot title.
`show_delta`	One of `"overlay"`, `"panel"`, or `"none"`. The default `"overlay"` draws proposal `delta_ic` values on a secondary y-axis. `"panel"` draws them in a lower panel. Logical values are accepted for compatibility: `TRUE` maps to `"overlay"` and `FALSE` maps to `"none"`.
`ic_limits`	Optional numeric vector of length 2 giving y-limits for the primary IC axis.
`delta_limits`	Optional numeric vector of length 2 giving y-limits for the `delta_ic` panel or secondary axis. The order is preserved, so `c(high, low)` reverses the axis and draws positive `delta_ic` values downward.
`xlab`, `ylab`	Axis labels.
`symbols`	Optional named vector or list of point symbols. Valid names are `accepted`, `rejected`, `error`, `baseline`, and `delta`.
`scales`	Optional named numeric vector or list of global scale multipliers. Valid names are `point`, `line`, and `text`.

`point_sizes`	Optional named numeric vector or list of point sizes. Valid names are `accepted`, `rejected`, `error`, `baseline`, and `delta`.
`line_widths`	Optional named numeric vector or list of line widths. Valid names are `accepted`, `rejected`, `error`, `running_best`, `delta`, and `zero`.
`text_sizes`	Optional named numeric vector or list of text sizes. Valid names are `main`, `axis`, `x_axis`, `y_axis`, `delta_axis`, `axis_label`, `annotation`, and `legend`.
`annotation`	Optional named numeric vector or list of annotation settings. Currently supports `baseline_label_offset`, the text offset for the baseline IC annotation passed to `graphics::text()` when `pos = 4`.
`legend`	Legend controls. Use `TRUE` or `FALSE` to show or hide the legend, or provide a named list with any of `show`, `position`, `inset`, `bty`, and `labels`. `position`, `inset`, `bty`, and `labels` are passed to `graphics::legend()` after validation. The default delta legend label is rendered as ΔIC.
`...`	Unused; included for S3 compatibility.

Value

Invisibly returns `x`.

Examples

```
search <- list(
  baseline_ic = -1000,
  IC_used = "GIC",
  model_fit_history = list(
    fits = list(
      list(
        step = 1, candidate_node = 42, regime_id = "1",
        ic = -1010, accepted = TRUE, delta_ic = 10
      ),
      list(
        step = 2, candidate_node = 57, regime_id = "2",
        ic = -1008, accepted = FALSE, delta_ic = -2
      )
    )
  )
)
class(search) <- c("bifrost_search", "list")
traj <- icTrajectory(search)

plot(traj)
plot(
  traj,
  delta_limits = c(15, -5),
  scales = c(point = 1.2, line = 1.1),
  point_sizes = c(rejected = 0.7),
  legend = list(position = "bottomleft")
)
```

plot.rateMap *Plot* rateMap *Objects*

Description

Render a computed "rateMap" object. The usual workflow is `x <- rateMap(...)` followed by `plot(x, ...)`.

Usage

```
## S3 method for class 'rateMap'
plot(
  x,
  value = "value",
  palette = NULL,
  reverse_palette = NULL,
  color_mode = NULL,
  n_categories = NULL,
  category_bin_method = NULL,
  category_breaks = NULL,
  category_labels = NULL,
  ncolors = NULL,
  legend_title = NULL,
  legend = NULL,
  fsize = NULL,
  tip_fsize = NULL,
  legend_fsize = NULL,
  ftype = NULL,
  show_tip_labels = TRUE,
  outline = FALSE,
  lwd = 3,
  type = c("phylogram", "fan", "arc"),
  mar = rep(0.3, 4),
  direction = "rightwards",
  offset = NULL,
  xlim = NULL,
  ylim = NULL,
  hold = TRUE,
  underscore = FALSE,
  arc_height = 2,
  legend_digits = NULL,
  ...
)
```

Arguments

x	An object of class "rateMap" returned by rateMap() or rateMapView().

value — Character column in the plotted summary table (x$intervals) to map to branch colors. The default "value" plots the central estimate chosen by rateMap(). When uncertainty = TRUE, useful alternatives include "mean", "median", "sd", "ci_width", "highest_density_interval_width", and "cv".

palette — Optional palette override used for this plot. This can be an hcl.colors() palette name, a vector of colors, or a palette function.

reverse_palette — Optional logical override for palette reversal used for this plot.

color_mode — Optional color-mode override. Use "continuous" for a numeric color ramp or "category" for ordered discrete rate categories. If NULL, the stored mode in x is used. Category mode draws one color per exact value or category bin; continuous mode draws a many-color ramp.

n_categories — Optional category-count override for color_mode = "category". This changes the target number of category bins, not the number of colors in continuous mode.

category_bin_method — Optional category-binning override for color_mode = "category". Use "pretty" for pretty breaks or "equal" for equal-width numeric intervals. Ignored when category_breaks is supplied.

category_breaks — Optional category-break override for color_mode = "category". Custom breaks override automatic bins. Category legends draw valid numeric interval breaks with proportional segment widths.

category_labels — Optional category-label override for color_mode = "category".

ncolors — Optional number of colors to use when recoloring this plot with color_mode = "continuous". If omitted, the stored x$ncolors value is used, falling back to 256 for older objects.

legend_title — Optional legend title override for this plot.

legend — Legend length. If NULL or TRUE, a layout-specific default is used. Set legend = FALSE to suppress the legend. Continuous legends are drawn with phytools::add.color.bar(); category legends are drawn by rateMap so bin labels can reflect rate categories and diagnostics.

fsize — Numeric font-size vector. The first element is passed as the tip label fsize to phytools::plotSimmap() and, when outline = TRUE, to phytools::plotTree(). The second element is used by the legend.

tip_fsize — Optional override for the first fsize element passed to the underlying tree plot.

legend_fsize — Optional override for the legend font size.

ftype — Font type. The first element is passed as ftype to phytools::plotSimmap() and, when outline = TRUE, to phytools::plotTree(). The second element is reserved for legends.

show_tip_labels — Logical; if FALSE, rateMap sets the tree-plot ftype to "off" before calling the underlying phytools plotter.

outline	Logical; if TRUE, draw a branch outline beneath the rate map. The outline pass is drawn with phytools::plotTree() before the colored phytools::plotSimmap() pass.
lwd	Branch and legend line widths. The first element is passed to phytools::plotSimmap() and, with + 2, to phytools::plotTree() for outlines. The second element is used for the legend.
type	Plot type: "phylogram", "fan", or "arc". This is passed to phytools::plotSimmap() for fan and arc layouts and to phytools::plotTree() for outline passes.
mar	Plot margins passed to phytools::plotSimmap() and, when outline = TRUE, to phytools::plotTree().
direction	Plotting direction for type = "phylogram"; passed to phytools::plotSimmap() and phytools::plotTree().
offset	Tip-label offset passed to phytools::plotSimmap() and, when outline = TRUE, to phytools::plotTree().
xlim, ylim	Optional plot limits passed to phytools::plotSimmap() and, when outline = TRUE, to phytools::plotTree().
hold	Logical controlling rateMap's device hold/flush guard. rateMap passes hold = FALSE to the internal phytools calls so the two-pass outline/color drawing is controlled in one place.
underscore	Logical; if FALSE, underscores in tip labels may be shown as spaces by the underlying phytools::plotSimmap() and phytools::plotTree() calls.
arc_height	Arc height passed through to phytools::plotSimmap() and, for outlines, phytools::plotTree() when type = "arc".
legend_digits	Optional number of digits for legend endpoint labels. If omitted, small-magnitude values use enough digits to avoid zero-valued legend endpoints.
...	Additional arguments are rejected. Include all display choices as named plot() arguments.

Details

The plot method uses phytools::plotSimmap() and draws either a continuous color-bar legend or a segmented rate-category color bar. The plotting controls intentionally mirror the phytools density-map plotting style for phylogram, fan, and arc layouts. In branch-summary category mode, near-zero or high-outlier rate diagnostics are drawn as special categories when rate-valued columns are plotted; these special categories do not consume positions in the ordered palette used for regular rate bins. When special categories are present, the category legend spans the full plotted value range and marks the diagnostic cutoff separating special and regular bins. When plotting non-rate columns such as "sd", diagnostic columns are preserved only as metadata with rate_flag_source provenance; special rate categories are not drawn for those non-rate values. The selected value column must contain finite values for every plotted interval; uncertainty columns such as "sd" may be all NA for single-fit objects and are rejected with a clear error.

The plot() method handles "phylogram", "fan", and "arc" layouts. legend controls the length of the color bar; set legend = FALSE to suppress it. legend_digits controls numeric endpoint labels and defaults to enough precision to avoid rounding small values to zero. Single fitted objects

should be converted explicitly with rateMap() before plotting, for example plot(rateMap(search_a), ...).

For fan and arc layouts, phytools resolves the getYmult() geometry helper from plotrix. bifrost imports that helper so installed-package plotting has the required dependency available without creating or modifying a getYmult binding in the user's global environment.

Relationship to phytools plotting arguments. plot.rateMap() keeps the tree-layout argument names close to phytools: type, fsize, ftype, lwd, mar, direction, offset, xlim, ylim, underscore, and arc_height are forwarded to phytools::plotSimmap() or phytools::plotTree() as described above. rateMap fixes phytools::plotSimmap() options such as colors, pts, node.numbers, add, and hold internally, because those are determined by the computed "rateMap" object and by the optional outline pass. palette, color_mode, n_categories, category_breaks, category_labels, legend_title, legend_digits, and the rate-flag display behavior are rateMap controls, not phytools arguments.

Value

Invisibly returns the plotted "rateMap" object.

References

Revell, L. J. (2013). Two new graphical methods for mapping trait evolution on phylogenies. *Methods in Ecology and Evolution*, 4, 754-759.

Revell, L. J. (2024). phytools 2.0: an updated R ecosystem for phylogenetic comparative methods (and other things). *PeerJ*, 12, e16505. doi:10.7717/peerj.16505

See Also

rateMap(), phytools::densityMap(), phytools::plotSimmap(), phytools::plotTree()

Examples

```
toy_tree <- ape::read.tree(text = "((a:1,b:1):1,c:2);")
toy_tree <- phytools::paintSubTree(
  toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
toy_fit <- list(
  variables = list(tree = toy_tree),
  param = c("0" = 0.1, "1" = 0.5)
)
rm_obj <- rateMap(toy_fit, log = FALSE, progress = FALSE)
plot(
  rm_obj,
  type = "arc",
```

```
  show_tip_labels = FALSE,
  legend = FALSE
)
```

`plot.rate_distribution_fit` *Plot A Fitted Rate Distribution*

Description

Plot a fitted candidate distribution from `fit_rate_distribution()`. The current method supports the Gumbel candidate used by Berv et al. (2026). It can overlay a pointwise bootstrap band and bootstrap density curves when `bootstrap_rate_distribution()` has already stored bootstrap draws on `x`.

Usage

```
## S3 method for class 'rate_distribution_fit'
plot(
  x,
  model = c("gumbel"),
  bootstrap = NULL,
  bootstrap_curves = 120L,
  col = "#b01f2e",
  hist_col = "grey90",
  band_alpha = 0.14,
  curve_alpha = 0.08,
  data_density = TRUE,
  rug = TRUE,
  breaks = "FD",
  main = NULL,
  xlab = NULL,
  ylab = "Density",
  ...
)
```

Arguments

`x`	A `rate_distribution_fit` object.
`model`	Distribution model to plot. Currently only `"gumbel"` is supported.
`bootstrap`	`NULL` or logical. If `NULL`, draw the bootstrap overlay when precomputed draws for `model` are available. If `TRUE`, require precomputed draws from `bootstrap_rate_distribution()`. If `FALSE`, draw no bootstrap overlay.
`bootstrap_curves`	Number of precomputed bootstrap density curves to draw over the band. Set to `0` to draw only the band.

col	Color for the fitted density, bootstrap band, and bootstrap curves.
hist_col	Histogram fill color. A vector of colors is passed through to graphics::hist() and can color bins individually.
band_alpha, curve_alpha	Alpha values for the bootstrap band and individual bootstrap curves.
data_density	Logical; overlay the empirical kernel density.
rug	Logical; add a rug for the fitted data values.
breaks	Histogram breaks passed to graphics::hist().
main, xlab, ylab	Plot labels.
...	Additional arguments passed to graphics::hist().

Value

Invisibly returns a list with the x grid, fitted density, bootstrap densities, bootstrap band, and plotted parameter table.

Examples

```
if (requireNamespace("univariateML", quietly = TRUE) &&
    requireNamespace("evd", quietly = TRUE)) {
  fit <- fit_rate_distribution(c(1, 1.4, 1.9, 2.8, 4.1, 6.5), models = "gumbel")
  fit <- bootstrap_rate_distribution(fit, model = "gumbel", reps = 25, seed = 1)
  plot(fit, bootstrap_curves = 5)
}
```

---

plot.regime_correlation_pca

Plot Regime Correlation PCA Results

Description

Draw compact PCA views from regime_correlation_pca(). The default "loadings" view reconstructs upper-triangle loading vectors as symmetric trait-by-trait heatmaps, matching the Supplementary Figure 4B-style interpretation of correlation-structure axes.

Usage

```
## S3 method for class 'regime_correlation_pca'
plot(
  x,
  type = c("loadings", "scores", "variance"),
  components = 1:4,
  pc_x = 1L,
  pc_y = 2L,
  palette = NULL,
  main = NULL,
```

```
  cluster = c("none", "global", "local"),
  heatmap_engine = c("auto", "base", "ComplexHeatmap"),
  show_dendrogram = TRUE,
  show_legend = TRUE,
  cex.axis = 0.62,
  ...
)
```

Arguments

x	A regime_correlation_pca object.
type	Plot type: "loadings", "scores", or "variance".
components	Principal components to draw for loading or variance plots. Supply numeric indices or names such as "PC1". By default, the first four available components, or all components when fewer exist, are drawn.
pc_x, pc_y	Components for the score scatter plot. The default uses PC1 and PC2 when available, or PC1 for both axes when only one component exists.
palette	Optional color vector for loading heatmaps.
main	Optional plot title. For loading plots with multiple components, supply one title per component.
cluster	Loading-heatmap clustering style: "none" preserves trait order, "global" reuses the first selected PC's clustered row/column order for all panels, and "local" reclusters each PC panel.
heatmap_engine	Loading-heatmap renderer. "auto" uses ComplexHeatmap for clustered heatmaps when that optional package is installed and otherwise falls back to base graphics.
show_dendrogram, show_legend	Logical controls used by the optional ComplexHeatmap renderer.
cex.axis	Axis label size for loading heatmaps.
...	Additional graphical parameters passed to base plotting functions.

Value

Invisibly returns x.

---

plot.regime_integration_relationships

Plot Regime Rate-Integration Relationships

Description

Draw Supplementary Figure 4A-style panels from regime_integration_relationships().

Usage

```
## S3 method for class 'regime_integration_relationships'
plot(
  x,
  panel = c("both", "variance", "correlation"),
  main = NULL,
  xlab = NULL,
  ylab = "Log inferred regime rate",
  point_alpha = 0.45,
  variance_col = "#4C78A8",
  correlation_col = "#D55E00",
  variance_line_col = "#1B4F72",
  correlation_line_col = "#8A3A00",
  ribbon_col = "grey70",
  pch = 21,
  lwd = 2,
  ...
)
```

Arguments

`x`	A `regime_integration_relationships` object.
`panel`	Which panel to draw: `"both"`, `"variance"`, or `"correlation"`.
`main`	Optional panel title. For `panel = "both"`, supply a length-two vector to title the variance and correlation panels separately.
`xlab`, `ylab`	Optional axis labels.
`point_alpha`	Point fill transparency.
`variance_col`, `correlation_col`	Point fill colors for the two panels.
`variance_line_col`, `correlation_line_col`	Curve colors for the two panels.
`ribbon_col`	Ribbon color.
`pch`, `lwd`	Base plotting symbol and line width.
`...`	Additional graphical parameters passed to `graphics::plot()`.

Value

Invisibly returns `x`.

`plot.regime_module_diagnostics` *Plot Regime Module Diagnostics*

Description

Draw scatterplots comparing selected PCA scores with selected module summaries from `regime_module_diagnostics()`.

Usage

```
## S3 method for class 'regime_module_diagnostics'
plot(
  x,
  pc,
  comparison,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  point_col = "#4C78A8",
  line_col = "#1B4F72",
  ribbon_col = "grey70",
  point_alpha = 0.5,
  pch = 21,
  lwd = 2,
  n_boot = 1000,
  ci_level = 0.99,
  seed = 1L,
  ...
)
```

Arguments

`x`	A `regime_module_diagnostics` object.
`pc`	Principal component name(s), such as `"PC1"`.
`comparison`	Module comparison name(s).
`main`	Optional panel title(s).
`xlab, ylab`	Optional axis labels. Recycled across panels.
`point_col, line_col`	Point fill and fitted-line colors.
`ribbon_col`	Ribbon color for bootstrap confidence intervals.
`point_alpha, pch, lwd`	Graphical controls.
`n_boot`	Number of bootstrap replicates for confidence ribbons. Set to `0` to omit the ribbon and draw only the fitted line.

ci_level	Confidence level for bootstrap ribbons.
seed	Optional random seed for reproducible bootstrap curves.
...	Additional graphical parameters passed to graphics::plot().

Value

Invisibly returns x.

---

plot.shift_magnitude_comparison

Plot One Shift-Magnitude Comparison

Description

Draw one density panel from a compare_shift_magnitudes() result. By default, the method reproduces the Berv et al. (2026) Supplementary Figure 5 scaling: each density integrates to that direction's observed frequency, so panel height reflects both shift magnitude and relative abundance. Set scale_by_frequency = FALSE for conventional unit-area densities.

Usage

```
## S3 method for class 'shift_magnitude_comparison'
plot(
  x,
  scale_by_frequency = TRUE,
  density_args = list(),
  colors = c(increase = "#377eb8", decrease = "#e41a1c"),
  fill_alpha = 0.22,
  lwd = 2,
  xlim = NULL,
  ylim = NULL,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  annotate = TRUE,
  show_modes = TRUE,
  show_sample_size = TRUE,
  show_test_label = TRUE,
  show_modal_ratio = TRUE,
  legend = TRUE,
  rug = FALSE,
  mar = c(4.5, 4.5, 2.4, 0.8),
  oma = c(0, 0, 0, 0),
  ...
)
```

Arguments

x	A single shift_magnitude_comparison object.
scale_by_frequency	Logical; multiply each density by its observed increase/decrease frequency.
density_args	Optional named list of additional arguments passed to stats::density().
colors	Named colors for "increase" and "decrease".
fill_alpha	Alpha value for density fills.
lwd	Line width for density curves.
xlim, ylim	Optional axis limits.
xlab, ylab, main	Plot labels.
annotate	Logical; draw panel annotations. This is a master switch for show_sample_size, show_test_label, and show_modal_ratio.
show_modes	Logical; draw dashed vertical lines at the KDE modes for increase and decrease magnitudes.
show_sample_size	Logical; include increase/decrease sample sizes in panel annotations when annotate = TRUE.
show_test_label	Logical; include KS statistics and p-values in panel annotations when annotate = TRUE and a KS test is available.
show_modal_ratio	Logical; include the back-transformed modal ratio in panel annotations when annotate = TRUE and it is finite.
legend	Logical; draw an increase/decrease legend.
rug	Logical; add rugs for the transformed values.
mar, oma	Graphical margins passed to graphics::par().
...	Additional graphical parameters passed to graphics::plot().

Value

Invisibly returns a list with the comparison object, summary tables, density estimates, and plot settings.

Examples

```
transitions <- data.frame(
  rate_change = c("increase", "increase", "increase", "decrease", "decrease", "decrease"),
   rate_delta = c(3, 4, 5, -1, -1.5, -2)
)
comparison <- compare_shift_magnitudes(transitions, ks_reps = 99)
plot(comparison)
```

plot.shift_magnitude_comparison_set

Plot A Shift-Magnitude Comparison Set

Description

Comparison sets contain multiple named comparisons. Select one named comparison, for example `x[["GIC"]]`, before plotting.

Usage

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

Arguments

`x`	A `shift_magnitude_comparison_set` object.
`...`	Ignored.

Value

Stops with an informative message.

plot.shift_magnitude_counts

Plot One Shift Count Or Frequency Comparison

Description

Summarize inferred rate increases and decreases for one set of search outputs. The default paired frequency plot mirrors the count/frequency comparison used for Berv et al. (2026) Supplementary Figure 5.

Usage

```
## S3 method for class 'shift_magnitude_counts'
plot(
  x,
  statistic = c("frequency", "count"),
  colors = c(increase = "#377eb8", decrease = "#e41a1c"),
  point_cex = 1.15,
  line_col = grDevices::adjustcolor("grey35", alpha.f = 0.35),
  boxplot = TRUE,
  paired_lines = TRUE,
  paired_test = TRUE,
```

```
  reference_line = NULL,
  ylim = NULL,
  ylab = NULL,
  main = NULL,
  mar = c(4.5, 4.2, 2.4, 0.8),
  oma = c(0, 0, 0, 0),
  ...
)
```

Arguments

x	A shift_magnitude_counts object.
statistic	Plot "frequency" or raw "count" values.
colors	Named colors for "increase" and "decrease".
point_cex	Point size for run-level observations.
line_col	Color for paired run-level segments.
boxplot	Logical; overlay compact boxplots.
paired_lines	Logical; draw paired decrease-to-increase segments for each run.
paired_test	Logical; annotate the plot with a paired Wilcoxon test when at least two runs are available.
reference_line	Optional horizontal reference line. Defaults to 0.5 for frequencies and no line for counts.
ylim	Optional common y-axis limits.
ylab, main	Plot labels.
mar, oma	Graphical margins passed to graphics::par().
...	Additional graphical parameters passed to graphics::plot().

Value

Invisibly returns a list with the count data, paired tests, and plot settings.

Examples

```
transitions <- data.frame(
  rate_change = c("root", "increase", "increase", "decrease", "decrease"),
  rate_delta = c(NA, 3, 4, -1, -2)
)
counts <- shift_magnitude_counts(list(run1 = transitions, run2 = transitions))
plot(counts)
```

plot.shift_magnitude_count_set *Plot A Shift-Magnitude Count Set*

Description

Count sets contain multiple named count tables. Select one named count table, for example `x[["GIC"]]`, before plotting.

Usage

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

Arguments

`x`	A `shift_magnitude_count_set` object.
`...`	Ignored.

Value

Stops with an informative message.

print.bifrost_search *Print method for* `bifrost` *search results*

Description

Prints a compact summary of a completed `bifrost` search, including the baseline and optimal information criterion (IC) values, the inferred shift node set, key search settings, and (when present) optional diagnostics such as IC-history and IC-weight support.

Usage

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

Arguments

`x`	A `bifrost_search` object returned by `searchOptimalConfiguration()`.
`...`	Unused (S3 compatibility).

Value

Invisibly returns `x`. Called for its printing side effects.

`print.bifrost_search_tuning_grid`
Print method for fixed-IC search tuning grids

Description

Print method for fixed-IC search tuning grids

Usage

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

Arguments

`x`	A `bifrost_search_tuning_grid` object returned by `runSearchTuningGrid()`.
`...`	Unused (S3 compatibility).

Value

Invisibly returns `x`. Called for its printing side effects.

See Also

`runSearchTuningGrid()`, `selectTunedSearchParameters()`

`print.bifrost_search_tuning_selection`
Print method for tuned search-parameter selections

Description

Print method for tuned search-parameter selections

Usage

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

Arguments

`x`	A `bifrost_search_tuning_selection` object returned by `selectTunedSearchParameters()`.
`...`	Unused (S3 compatibility).

Value

Invisibly returns `x`. Called for its printing side effects.

See Also

`selectTunedSearchParameters()`, `runSearchTuningGrid()`

`print.bifrost_shift_recovery_evaluation`
Print method for shift-recovery evaluations

Description

Print method for shift-recovery evaluations

Usage

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

Arguments

`x`	A `bifrost_shift_recovery_evaluation` object returned by `evaluateShiftRecovery()`.
`...`	Unused (S3 compatibility).

Value

Invisibly returns `x`. Called for its printing side effects.

See Also

`evaluateShiftRecovery()`

`print.bifrost_simulation_study`
Print method for bifrost simulation studies

Description

Print a compact summary of a `bifrost_simulation_study`, including the study type, generating scenario, replicate counts, and the main false-positive or recovery summaries attached to the study object.

Usage

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

Arguments

x	A bifrost_simulation_study object returned by runFalsePositiveSimulationStudy() or runShiftRecoverySimulationStudy().
...	Unused (S3 compatibility).

Value

Invisibly returns x. Called for its printing side effects.

See Also

runFalsePositiveSimulationStudy(), runShiftRecoverySimulationStudy()

Examples

```
study <- structure(
  list(
    study_type = "false_positive",
    generating_scenario = "null",
    study_summary = list(
      n_replicates = 2L,
      n_completed = 2L,
      n_failed = 0L,
      mean_false_positive_rate = 0,
      median_false_positive_rate = 0
    )
  ),
  class = c("bifrost_simulation_study", "list")
)
study
```

---

print.bifrost_simulation_template

Print method for bifrost simulation templates

Description

Print a compact summary of a bifrost_simulation_template, including the aligned tree size, response/predictor structure, global calibration formula, downstream simulation-search formula, and the empirical covariance summaries used to generate simulation replicates.

Usage

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

Arguments

x	A bifrost_simulation_template object returned by createSimulationTemplate().
...	Unused (S3 compatibility).

Value

Invisibly returns x. Called for its printing side effects.

See Also

createSimulationTemplate()

Examples

```
set.seed(1)
tr <- ape::rtree(12)
X <- matrix(rnorm(12 * 2), ncol = 2)
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")
tmpl
```

---

print.rateMap *Print a* rateMap *Object*

Description

Print a concise summary of a "rateMap" object, including the number of fits, summary mode, target/check mode, weighting mode, uncertainty status, color mode, plotted value, value range, rate-flag diagnostics when active, and uncertainty ranges when available.

Usage

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

Arguments

x	An object of class "rateMap" returned by rateMap().
...	Ignored.

Value

Invisibly returns x.

rateMap *Compute Branchwise Rate Maps Across Runs*

Description

Summarize fitted regime-specific rates across a list of stochastic-map-aware model fits or completed `bifrost_search` results. By default, `rateMap()` follows `bifrost`'s branch-level framing: the first retained tree is used as the plotting scaffold, all retained trees must match in topology and branch lengths, each branch receives one summarized log-rate, and runs are averaged with equal weight.

Usage

```
rateMap(
  fits,
  weights = c("equal", "ic"),
  uncertainty = FALSE,
  summary = c("branch", "interval"),
  log = TRUE,
  value_summary = c("mean", "median"),
  target_tree = NULL,
  workers = NULL,
  progress = TRUE,
  control = rateMapControl()
)
```

Arguments

`fits`	A completed run, fitted model object, or non-empty list of these objects. Supported shapes are `bifrost_search` objects, `mvgls` objects, `list(model = <mvgls>)`, and scratch-style lists with `variables$tree` plus `param`. Single supported objects are wrapped automatically as a convenience for one-fit plotting and inspection. The default extractor requires a mapped tree and a named, finite, positive rate for each mapped regime. A baseline-only BM search has `model_no_uncertainty$param = NA` and cannot currently be passed directly to `rateMap()` with the default extractor. Its fitted global covariance is available in `search$VCVs[["0"]]`. If invalid fits are omitted from a multi-run map, report the omission because the map summarizes only the retained fits.
`weights`	Fit-level weighting mode. `"equal"` gives every retained fit equal weight. `"ic"` computes standard IC weights from `optimal_ic` and requires all retained fits to have the same `IC_used`. A numeric vector is also accepted and treated as custom fit weights.
`uncertainty`	Logical; if `TRUE`, compute and return across-fit uncertainty summaries for every summary row: whole branches when `summary = "branch"` and depth-grid intervals when `summary = "interval"`.

summary	Character; "branch" computes one length-weighted value per edge, while "interval" slices branches on a global depth grid.
log	Logical; if TRUE (the default), transform extracted rate parameters with log() before branch, interval, and across-fit averaging. Weighted means are then mean log-rates, equivalent to the log of weighted geometric mean rates. Set log = FALSE to summarize rates in their original units.
value_summary	Character; central estimate stored in the summary table column intervals$value. "mean" uses the weighted mean. "median" uses the weighted median. When log = TRUE, both summaries are computed on the log-rate scale.
target_tree	Optional explicit target tree used as the summary and plotting scaffold. This may be any target or summary tree with the same topology and tip labels as the inputs. It does not need to contain stochastic maps because rateMap() replaces maps with display maps in the returned object. Branch lengths in target_tree define the geometry of the returned and plotted tree.
workers	Optional number of future workers to use. If NULL, the current future::plan() is used as-is.
progress	Logical; if TRUE, display a text progress bar via progressr::with_progress().
control	A "rateMap_control" object from rateMapControl(), or a named list of advanced control options.

Details

Everyday workflow. rateMap() is a compute function: call it to build a reusable "rateMap" object, then call plot(x, ...) to choose what to display. Common compute controls are fits, weights, uncertainty, summary, log, and value_summary. Common display controls belong in plot() or rateMapView(), including value, type, palette, color_mode, n_categories, show_tip_labels, and legend sizing. Controls such as target, check, res, tree_fun, param_fun, and the future_* arguments live in rateMapControl() because they are mainly advanced tools for same-topology tree samples, interval summaries, custom fit objects, and larger run sets.

Algorithmic provenance. rateMap() is inspired by phytools::densityMap(), which summarizes a set of stochastic maps by slicing mapped branches on a shared depth grid and coloring a SIMMAP-style tree by an averaged branchwise quantity. Here the averaged quantity is not a posterior probability of a mapped state; instead, each mapped state is translated through the corresponding fitted regime-rate parameter from each run. The plotting interface likewise follows the phytools density-map family by returning a colored SIMMAP tree and drawing it with phytools::plotSimmap() plus either a segmented rate-category color bar or a continuous color-bar legend.

Although rateMap() is designed for summarizing multiple fitted maps, it also accepts a single completed bifrost search or supported fit. Single-fit input is a convenience for inspecting one fitted model before scaling up to multi-run summaries.

rateMap() can also summarize same-topology posterior or sensitivity samples where branch lengths differ. In that case, usually supply an explicit target_tree and use control = list(check = "topology"). The target can be any tree with the same topology and tip labels as the inputs; an MCC tree is only one possible choice. The target tree supplies the plotted topology and branch lengths, while rates are matched from each input tree by descendant-tip clade keys rather than by edge order.

Summary modes. With the default `summary = "branch"` and `log = TRUE`, each edge receives one length-weighted average log-rate from each run before the across-run summary is computed. This is the natural default for `bifrost` searches because shifts are placed at nodes, so a `bifrost` branch is not expected to change regimes internally. With `summary = "interval"`, each target-tree branch is subdivided by the global depth grid controlled by `control = list(res = ...)`. If source branch lengths differ from the target branch length, source stochastic-map segments are projected onto target intervals by relative position along the matched branch. Interval mode is useful for general stochastic maps that can genuinely change state along a branch.

Log-rate averaging. When `log = TRUE`, rate parameters are transformed before branch-level, interval-level, and across-fit summaries are computed. With weights `w_i`, the default plotted mean is therefore `sum(w_i * log(rate_i))`, which is the log of a weighted geometric mean. It is not `log(sum(w_i * rate_i))`, the log of a weighted arithmetic mean. Use `log = FALSE` when downstream interpretation requires arithmetic summaries on the original rate scale. Raw fitted rate parameters must be strictly positive in either mode. Negative plotted values are therefore valid in log-rate maps whenever the positive raw fitted rates are less than one.

Tree checks and targets. In `rateMapControl()`, `check = TRUE` is equivalent to `check = "full"` and requires topology and branch lengths to match the target tree. Use `check = "topology"` when all inputs have the same topology and tip labels but may have different branch lengths. `check = FALSE` or `check = "none"` skips the upfront `ape::all.equal.phylo()` check, but every target branch must still be recoverable in every input tree by descendant-tip set. This function is not a mixed-topology posterior summarizer; clades absent from an input tree are treated as an error rather than being marginalized over topology.

When `target_tree` is supplied, it is used directly as the plotting scaffold and need not contain SIMMAP maps. When `target_tree = NULL`, `rateMapControl(target = "first")` uses the first retained input tree, and `rateMapControl(target = "mcc")` chooses the retained input tree with the highest sum of log clade credibilities. For truly same-topology inputs, the MCC score is usually tied, so `"mcc"` commonly resolves to the first retained tree. MCC target selection does not make `rateMap()` a mixed-topology summarizer: every target branch must still be present in every retained input. If you already have a preferred consensus, chronogram, MCC, maximum-likelihood, or otherwise curated target tree, pass it with `target_tree`.

Fit weights. `weights = "equal"` assigns the same weight to each retained fit. `weights = "ic"` computes standard information-criterion weights from each retained fit's `optimal_ic`, requiring all retained fits to share the same non-missing `IC_used`. A numeric `weights` vector can be supplied for custom weighting. Weights are subset to retained fits after `rateMapControl(na_action = "omit")` and are normalized to sum to one. IC weights are descriptive fit-level weights for comparable retained searches; they do not choose a formal threshold or make incomparable searches comparable. Weights are assigned to supplied fits, not unique shift configurations. If the same or a similar configuration appears in several fits, its combined contribution reflects each of those fits.

Uncertainty summaries. The returned `intervals` data frame is the plotted summary table. With `summary = "branch"`, it has one row per branch. With `summary = "interval"`, it has one row per plotted depth-grid interval. When `uncertainty = TRUE`, this table includes across-fit summaries for each plotted row: weighted mean, weighted median, weighted standard deviation, quantiles, highest-density interval bounds, quantile and highest-density interval widths, coefficient of variation, and the number of finite run-level values. The run-level values are also returned in `run_values` as one matrix per edge. These are weighted empirical summaries of run-level values, not posterior uncertainty or model-internal uncertainty unless the retained inputs themselves have that interpretation. For posterior tree samples, use `weights = "equal"` if each retained run represents one pos-

terior draw. value_summary controls whether the plotted value column uses the weighted mean or weighted median. These summaries are computed on the log-rate scale when log = TRUE. Highest-density intervals use the same shortest empirical interval calculation for both equal and unequal fit weights. For weights = "equal", sd is the ordinary sample standard deviation of retained run-level values. For unequal weights, sd is the square root of the normalized weighted variance, sum(w * (x - mu)^2), with weights normalized to sum to one.

Display views. Returned objects include a default category-style display mapping so they can be plotted immediately. Palette, category-bin, legend title, and alternative-value choices are intentionally controlled by plot(x, ...) or rateMapView(), not by rateMap(). For a single bifrost search with log = FALSE, category display is the closest formal analogue to the illustrative generateViridisColorScale() plot. For multi-run summaries, displayed categories are bins for summarized branch values; they should not be read as newly inferred bifrost regimes. When summary = "branch" and color_mode = "category", the rate_categories table also reports branch-level summaries for the plotted values assigned to each bin. These summaries are recomputed by rateMapView() or plot() whenever the plotted value or category breaks change. They are not computed for summary = "interval" because interval rows depend on the plotting grid rather than on discrete biological branch units.

Rate diagnostics. Branch-summary maps record the fitted-rate range by default, without flagging near-zero or high-outlier rates. Use rateMapRateFlags() through rateMapControl() to apply a manual zero_floor or the data-driven tail_cluster diagnostic. Flags add metadata and optional display categories without removing branches or changing their summarized values. In category plots, flagged branches use special colors outside the ordered palette, and the legend marks the cutoff between special and regular categories. Set rate_flags = NULL in rateMapControl() to disable diagnostics. See rateMapRateFlags() for detection methods and settings, and plot.rateMap() for display behavior.

Value

An object of class "rateMap" with components:

tree A SIMMAP-style tree whose mapped segments encode color-bin indices in continuous mode, or named rate categories in category mode.

cols The resolved color palette. In continuous mode this has length ncolors; in category mode it has one color per exact value or category bin.

lims Numeric length-2 vector giving the plotted value range.

breaks Numeric vector of palette bin boundaries in continuous mode, or category boundaries/values in category mode.

values List of plotted-row central values by edge before color binning.

intervals Plotted summary table. With summary = "branch", this has one row per branch. With summary = "interval", this has one row per plotted depth-grid interval. When branch-level rate diagnostics are enabled, this table also includes rate_for_flagging, rate_flag, rate_flag_source, is_near_zero, and is_high_outlier.

rate_categories Data frame describing discrete rate categories when color_mode = "category"; otherwise NULL. With summary = "branch", this table also includes bin-level summaries of the plotted branch values, including n_branches, value_mean, value_median, value_min, value_max, value_sd, and total_branch_length.

`run_values` When `uncertainty = TRUE`, list of numeric matrices containing run-level values for each edge. Matrix rows match the plotted rows for that edge and columns are retained fits. Otherwise `NULL`.

`clade_key` Character descendant-tip key for each target-tree edge.

`edge_matches` Integer matrix mapping target-tree edge rows to matched source-tree edge rows for each retained fit.

`summary` The summary mode used, `"interval"` or `"branch"`.

`uncertainty` Logical indicating whether uncertainty summaries were computed.

`value_summary` Central estimate used for the plotted summary table column `intervals$value`.

`quantile_probs` Quantile probabilities used for uncertainty summaries.

`highest_density_interval_prob` Highest-density interval mass used for uncertainty summaries.

`rate_diagnostics` List summarizing rate-flag settings, counts, detected tail cutoffs, and fold-rate ranges with and without flagged branches.

`rate_flags` The normalized `"rateMap_rate_flags"` control object used for rate diagnostics.

`rate_flag_source` Character name of the rate-valued column used to compute or preserve `rate_flag` metadata, or `NA` when diagnostics are disabled.

`plot_value` Current interval column mapped to branch colors.

`target` Target-tree selection mode used.

`check` Tree compatibility check mode used.

`weights` Normalized fit weights used for aggregation.

`weight_mode` Weighting mode used: `"equal"`, `"ic"`, or `"custom"`.

`weight_table` Data frame linking retained input indices, weights, and IC values when available.

`palette` Original palette specification.

`reverse_palette` Logical indicating whether the palette was reversed.

`ncolors` Stored continuous-ramp resolution used when recoloring with `color_mode = "continuous"` and no explicit `ncolors`.

`color_mode` Coloring mode used for the current tree.

`n_categories` Category count target used when `color_mode = "category"`.

`category_breaks` Category breaks or exact category values used when `color_mode = "category"`.

`category_labels` Category labels used when `color_mode = "category"`.

`category_bin_method` Automatic category-binning method used when `color_mode = "category"` and `category_breaks = NULL`.

`title` Legend title used for plotting.

`n_fits` Number of fits used after validation or omission.

`omitted` Integer indices of omitted fits when `na_action = "omit"`.

References

Paradis, E., and Schliep, K. (2019). ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35, 526-528. doi:10.1093/bioinformatics/bty633

Clavel, J., Aristide, L., and Morlon, H. (2019). A penalized likelihood framework for high-dimensional phylogenetic comparative methods and an application to new-world monkeys brain evolution. *Systematic Biology*, 68, 93-116. doi:10.1093/sysbio/syy045

Revell, L. J. (2013). Two new graphical methods for mapping trait evolution on phylogenies. *Methods in Ecology and Evolution*, 4, 754-759.

Revell, L. J. (2024). phytools 2.0: an updated R ecosystem for phylogenetic comparative methods (and other things). *PeerJ*, 12, e16505. doi:10.7717/peerj.16505

See Also

`plot.rateMap()`, `rateMapView()`, `rateMapControl()`, `phytools::densityMap()`, `phytools::plotSimmap()`

Examples

```
## Not run:
# A list of completed bifrost searches can be summarized directly:
rm_obj <- rateMap(list(search_a, search_b, search_c))
plot(rm_obj, type = "arc", show_tip_labels = FALSE)

# A single completed bifrost search can be inspected with the same API:
one_search_map <- rateMap(search_a)
plot(one_search_map, type = "arc", show_tip_labels = FALSE)

# Scratch-style lists remain supported:
scratch_fit <- list(
  variables = list(tree = mapped_tree),
  param = c("0" = 0.12, "1" = 0.45)
)
rateMap(list(scratch_fit))

# Use bifrost model-level IC weights across comparable sensitivity runs:
rm_ic <- rateMap(list(search_a, search_b), weights = "ic")

# Branch-level, discrete-category maps are the bifrost default:
branch_rates <- rateMap(list(search_a, search_b))
branch_rates$rate_categories

# To mimic the old jaw-shape preview style for a single search, use raw
# fitted rates and category colors:
jaw_view <- rateMapView(rateMap(search_a, log = FALSE), palette = viridis::viridis)

# Category bins use pretty breaks by default. Use equal-width bins or custom
# boundaries when those are easier to compare across figures:
equal_bin_rates <- rateMapView(
  branch_rates,
  n_categories = 5,
  category_bin_method = "equal"
```

```
)
custom_bin_rates <- rateMapView(
  branch_rates,
  category_breaks = c(-4, -2, 0, 2),
  category_labels = c("slow", "middle", "fast")
)

# Continuous interval maps remain available for stochastic maps with
# along-branch changes:
interval_rates <- rateMap(
  list(search_a, search_b),
  summary = "interval",
  control = list(res = 200)
)
plot(interval_rates, color_mode = "continuous")

# Custom fit weights are normalized internally:
custom_weighted <- rateMap(
  list(search_a, search_b, search_c),
  weights = c(2, 1, 1),
  summary = "branch"
)

# Posterior trees with the same topology but different branch lengths can be
# summarized on any explicit target or summary tree:
posterior_target_rates <- rateMap(
  posterior_fit_list,
  target_tree = summary_tree,
  summary = "branch",
  weights = "equal",
  uncertainty = TRUE,
  control = list(check = "topology")
)
plot(posterior_target_rates, value = "sd", type = "arc")

# Or choose the retained input tree with the highest summed log clade
# credibility as the plotting scaffold:
posterior_mcc_rates <- rateMap(
  posterior_fit_list,
  summary = "branch",
  control = list(check = "topology", target = "mcc")
)

# Large sensitivity sets can be explicitly subsampled before plotting.
# By default, rates are mapped on the log scale:
set.seed(1)
idx <- sample(seq_along(fit_list), size = 1000)
rm_sub <- rateMap(
  fit_list[idx],
  workers = 8,
  control = list(res = 100, future_strategy = "multisession")
)
```

```
# Use log = FALSE only when a raw-rate scale is preferred:
rm_raw <- rateMap(fit_list[idx], log = FALSE)
plot(
  rm_sub,
  type = "arc",
  show_tip_labels = FALSE,
  lwd = 1,
  palette = c("lightblue", "blue", "pink", "red")
)

## End(Not run)
```

`rateMapControl` *Control Advanced* `rateMap()` *Options*

Description

Build a control object for less common `rateMap()` settings. These options are useful for interval maps, same-topology tree samples, custom fit object shapes, invalid-fit handling, and future-based parallel scheduling.

Usage

```
rateMapControl(
  res = 100,
  check = TRUE,
  target = c("first", "mcc"),
  tree_fun = NULL,
  param_fun = NULL,
  na_action = c("error", "omit"),
  rate_flags = rateMapRateFlags(),
  quantile_probs = c(0.025, 0.975),
  highest_density_interval_prob = 0.95,
  future_strategy = c("multisession", "multicore"),
  future_seed = FALSE,
  future_scheduling = 1,
  future_chunk_size = NULL
)
```

Arguments

`res`	Integer resolution of the global depth grid used to subdivide target-tree branches when `summary = "interval"`. Ignored by `summary = "branch"`.
`check`	Logical or character check mode. `TRUE` or `"full"` verifies that extracted trees and the target tree match in topology and branch lengths. `"topology"` verifies matching topology/tip labels while allowing branch lengths to differ. `FALSE` or `"none"` skips the upfront `ape::all.equal.phylo()` check, but target branches still must be matchable by descendant-tip sets.

target — Character target-tree selection when target_tree = NULL in rateMap(). "first" uses the first retained input tree. "mcc" chooses the retained input tree with the highest sum of log clade credibilities.

tree_fun — Optional function used to extract a mapped tree from each element of fits. If NULL, common bifrost and mvgls shapes are auto-detected. For "bifrost_search" objects, the default uses tree_no_uncertainty_untransformed to preserve the original branch-length scale; supply tree_fun explicitly to map a different tree field.

param_fun — Optional function used to extract a named numeric vector of state-specific fitted rates from each element of fits. If NULL, common bifrost and mvgls shapes are auto-detected.

na_action — What to do when a run has invalid parameters. "error" stops immediately. "omit" drops invalid runs before aggregation.

rate_flags — A "rateMap_rate_flags" object from rateMapRateFlags(), or a named list of rate-flagging options. Use NULL to disable rate diagnostics.

quantile_probs — Numeric length-2 vector of quantile probabilities to report when uncertainty = TRUE.

highest_density_interval_prob — Numeric scalar giving the highest-density interval mass to report when uncertainty = TRUE.

future_strategy — Future backend used only when workers is supplied to rateMap(). Must be one of "multisession" or "multicore".

future_seed — Seed control passed to future.apply::future_lapply().

future_scheduling — Scheduling control passed to future.apply::future_lapply().

future_chunk_size — Chunk size passed to future.apply::future_lapply().

Value

A "rateMap_control" object for the control argument of rateMap().

Examples

```
ctrl <- rateMapControl(
  res = 200,
  check = "topology",
  na_action = "omit",
  rate_flags = rateMapRateFlags(zero_floor = 1e-8)
)
ctrl[c("res", "check", "na_action")]
ctrl$rate_flags[c("method", "zero_floor")]
```

rateMapRateFlags *Control Near-Zero and Tail Rate Diagnostics for* rateMap()

Description

Build a rate-flagging control object for rateMapControl(). These options identify branch-rate summaries that are effectively zero, or optionally in a separated high-rate tail, without deleting or changing the fitted rates. Flags are reported in the returned intervals table and summarized in rate_diagnostics.

Usage

```
rateMapRateFlags(
  near_zero = NULL,
  high_outlier = FALSE,
  method = NULL,
  zero_floor = NULL,
  cluster_min_log_gap = log(10),
  cluster_min_fold_reduction = 10,
  cluster_max_tail_fraction = 0.4,
  cluster_min_flagged = 3,
  cluster_min_regular = 10,
  zero_label = "near-zero",
  high_label = "high-outlier",
  zero_color = "grey70",
  high_color = "black"
)
```

Arguments

near_zero	Logical or NULL. If TRUE, flag lower-tail or floor-level rates. The default NULL resolves to FALSE for method = "none" and TRUE for method = "floor" or method = "tail_cluster".
high_outlier	Logical; if TRUE, flag isolated upper-tail rates.
method	Optional detection method. NULL chooses "floor" when zero_floor is supplied and "none" otherwise. "floor" uses zero_floor for near-zero rates. "tail_cluster" uses an Otsu-style guarded two-class split of log rates to identify a separated lower-tail cluster, and, when high_outlier = TRUE, a separated upper-tail cluster. "none" computes diagnostics without adding special rate flags. Inactive switches are normalized to FALSE: method = "none" sets near_zero and high_outlier to FALSE, and method = "floor" sets high_outlier to FALSE.
zero_floor	Optional non-negative rate floor. When supplied with method = NULL, the method resolves to "floor". Finite rates less than or equal to this value are flagged as near-zero.

`cluster_min_log_gap`
: Minimum log-rate gap required between a cluster-defined tail and the remaining regular rates.

`cluster_min_fold_reduction`
: Minimum fold-range reduction required after separating a cluster-defined tail from the regular rates.

`cluster_max_tail_fraction`
: Maximum fraction of positive finite rates that can be assigned to a cluster-defined tail.

`cluster_min_flagged`
: Minimum number of branches required for a cluster-defined tail.

`cluster_min_regular`
: Minimum number of branches that must remain in the regular rate set after separating a cluster-defined tail.

`zero_label`, `high_label`
: Labels used for flagged display categories.

`zero_color`, `high_color`
: Colors used for flagged display categories in category mode.

Details

The `"tail_cluster"` method is an Otsu-style diagnostic adapted to sorted branch log rates. It evaluates two-class splits and keeps a tail only when guardrails for log-rate gap size, fold-range reduction, tail fraction, and minimum counts are all satisfied. It is display metadata, not data deletion, model correction, or formal threshold selection.

Value

A normalized `"rateMap_rate_flags"` object for `rateMapControl(rate_flags = )`.

References

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1), 62-66. doi:10.1109/TSMC.1979.4310076

Examples

```
floor_flags <- rateMapRateFlags(zero_floor = 1e-8)
floor_flags[c("method", "near_zero", "zero_floor")]

tail_flags <- rateMapRateFlags(
  method = "tail_cluster",
  high_outlier = TRUE
)
tail_flags[c("method", "near_zero", "high_outlier")]
```

rateMapView *Create a Display View of a* rateMap *Object*

Description

Recompute the display mapping for a computed "rateMap" object without drawing it. This is optional in the usual rateMap() then plot() workflow; use it when you want to save or inspect category tables, color bins, or an uncertainty-valued map object and then reuse the same display mapping in one or more plots. The numeric branch or interval summaries computed by rateMap() are not recomputed.

Usage

```
rateMapView(
  x,
  value = "value",
  palette = NULL,
  reverse_palette = NULL,
  color_mode = NULL,
  n_categories = NULL,
  category_bin_method = NULL,
  category_breaks = NULL,
  category_labels = NULL,
  ncolors = NULL,
  legend_title = NULL
)
```

Arguments

x	An object of class "rateMap" returned by rateMap().
value	Name of the numeric column in x$intervals to map to colors. The default, "value", uses the central summary computed by rateMap().
palette	Optional palette override. This can be an hcl.colors() palette name, a vector of colors, or a palette function.
reverse_palette	Optional logical override for palette reversal.
color_mode	Optional color-mode override. Use "category" for discrete ordered bins or "continuous" for a continuous ramp.
n_categories	Optional target category count for color_mode = "category".
category_bin_method	Optional automatic category-binning method: "pretty" or "equal".
category_breaks	Optional strictly increasing category boundaries. Category legends draw valid numeric interval breaks with proportional segment widths.

`category_labels`
: Optional labels for displayed categories.

`ncolors`
: Optional number of colors for continuous ramps. If omitted, the stored `x$ncolors` value is used, falling back to `256` for older objects.

`legend_title`
: Optional legend title stored on the returned object.

Value

A `"rateMap"` object with updated tree maps, color palette, `intervals$value`, `rate_categories`, and legend title. For branch-summary category views, `rate_categories` includes bin-level summaries of the plotted branch values and is recomputed whenever the view changes. For branch-summary maps with active rate diagnostics, diagnostic flags are recomputed for rate-valued views (`"value"`, `"mean"`, or `"median"`) and preserved as metadata for non-rate views such as `"sd"`. When preserved for a non-rate view, `rate_flag_source` identifies the rate-valued column that the flags classify; the flags do not classify the displayed uncertainty value. The selected `value` column must contain finite values for every plotted interval; uncertainty columns such as `"sd"` may be all `NA` for single-fit objects and are rejected with a clear error.

Examples

```
toy_tree <- ape::read.tree(text = "((a:1,b:1):1,c:2);")
toy_tree <- phytools::paintSubTree(
  toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
toy_fit <- list(
  variables = list(tree = toy_tree),
  param = c("0" = 0.1, "1" = 0.5)
)
rm_obj <- rateMap(toy_fit, log = FALSE, progress = FALSE)
display_obj <- rateMapView(
  rm_obj,
  palette = "Viridis",
  n_categories = 3,
  legend_title = "Fitted rate"
)
display_obj[c("color_mode", "category_labels", "title")]
```

---

regime_correlation_pca

Run PCA on Regime Correlation Structures

Description

Convert each regime covariance matrix to a correlation matrix, vectorize the upper-triangle off-diagonal entries, and run `stats::prcomp()` on the resulting regime-by-correlation matrix. When `use_correlation = TRUE`, the function warns if all retained covariance matrices are scalar-proportional to one another. That pattern is expected for proportional `search$VCVs` from the scalar `bifrost` search model and should not be interpreted as independent post-hoc covariance reconfiguration evidence.

Usage

```
regime_correlation_pca(
  x,
  use_correlation = TRUE,
  center = TRUE,
  scale. = TRUE,
  tip_counts = NULL,
  regime_ages = NULL,
  trait_labels = NULL,
  min_tips = NULL,
  ...
)
```

Arguments

`x`	A `regime_covariances` object returned by `fit_regime_covariances()` or a named list of covariance/correlation matrices. Matrices must contain finite entries and strictly positive diagonal variances.
`use_correlation`	Logical; if `TRUE`, convert each matrix with `stats::cov2cor()` before vectorizing.
`center, scale.`	Passed to `stats::prcomp()`. Defaults match the manuscript workflow.
`tip_counts`	Optional named numeric vector of tip counts.
`regime_ages`	Optional named numeric vector of regime ages.
`trait_labels`	Optional labels for the matrix rows/columns. Supply a named vector to map internal trait names to display labels, or an unnamed vector in matrix order. Resolved labels must be unique and not blank.
`min_tips`	Optional positive whole-number minimum tip count for inclusion when `tip_counts` are available. Matching the manuscript `min_n` convention, regimes are retained only when `tip_count > min_tips`. At least one non-missing tip count must be available when this filter is used.
`...`	Reserved for future extensions.

Value

An object of class `regime_correlation_pca` containing the `prcomp` object, vectorized input matrix, scores, loadings, variance explained, regime IDs, trait labels, optional diagnostics, and a `settings` list that records the PCA scaling, tip-count filter, and whether retained covariance matrices were scalar-proportional.

Examples

```
make_correlation <- function(r12, r13, r23) {
  matrix(
    c(1, r12, r13, r12, 1, r23, r13, r23, 1),
    nrow = 3,
    dimnames = list(c("bill", "wing", "tail"),
                    c("bill", "wing", "tail"))
  )
}
covariances <- list(
  r1 = make_correlation(0.1, 0.2, 0.3),
  r2 = make_correlation(0.2, 0.1, 0.4),
  r3 = make_correlation(-0.1, 0.3, 0.2),
  r4 = make_correlation(0.4, -0.2, 0.1)
)
pca <- regime_correlation_pca(covariances)
pca$variance_explained
```

`regime_integration_pgls` *Fit the Manuscript-style Regime Integration pGLS*

Description

Reproduce the representative pGLS test used in the post-hoc integration analysis: log regime rate is modeled as a function of log post-hoc mean variance and Fisher-Z transformed mean absolute trait correlation on a collapsed regime phylogeny.

Usage

```
regime_integration_pgls(
  summary_data,
  search = NULL,
  tree = NULL,
  model = "BM",
  min_tips = NULL,
  ...
)
```

Arguments

`summary_data`	A data frame from `summarize_regime_covariances()` or a manuscript-compatible `vars_cors` table with columns `rate`, `vars`, `corrs`, and `State`. Regime IDs must be unique and non-empty.
`search`	A `bifrost_search` object containing the mapped regime tree.
`tree`	Optional SIMMAP-style mapped tree. Ignored when `search` is supplied.

model	Evolutionary model passed to phylolm::phylolm(). Defaults to "BM", matching the manuscript.
min_tips	Optional minimum tip count for downstream inclusion when summary_data contains a tip_count column. Downstream summaries are retained when tip_count >= min_tips; this differs from the strict PCA manuscript filter used by regime_correlation_pca().
...	Additional arguments passed to phylolm::phylolm().

Details

The collapse step follows the manuscript implementation: a monophyletic regime is represented by one collapsed tip, whereas all tips assigned to a nonmonophyletic regime are removed. When removals occur, the function emits one warning listing every dropped regime ID. The collapse stops if regime relabeling would create duplicated output tip labels. Standardization is computed across all summary rows surviving the optional min_tips filter before rows absent from the collapsed regime phylogeny are dropped. This intentional ordering matches manuscript preprocessing.

Value

A phylolm fit.

Examples

```
if (requireNamespace("phylolm", quietly = TRUE)) {
  tree <- ape::read.tree(text = paste0(
    "((((a:1,b:1):1,(c:1,d:1):1):1,",
    "((e:1,f:1):1,(g:1,h:1):1):1):1,(i:1,j:1):3);"
  ))
  tree <- phytools::paintSubTree(
    tree, node = ape::Ntip(tree) + 1L, state = "root"
  )
  tip_pairs <- list(
    r1 = c("a", "b"), r2 = c("c", "d"), r3 = c("e", "f"),
    r4 = c("g", "h"), r5 = c("i", "j")
  )
  for (regime in names(tip_pairs)) {
    tree <- phytools::paintSubTree(
      tree,
      node = ape::getMRCA(tree, tip_pairs[[regime]]),
      state = regime
    )
  }
  summary_data <- data.frame(
    regime = names(tip_pairs),
    rate = c(0.8, 1.1, 1.7, 2.2, 3.4),
    mean_variance = c(0.6, 0.9, 1.3, 2.0, 2.7),
    mean_abs_correlation = c(0.10, 0.18, 0.25, 0.32, 0.41)
  )
  fit <- regime_integration_pgls(summary_data, tree = tree)
  stats::coef(fit)
}
```

`regime_integration_relationships` *Analyze Rate-Integration Relationships Across Regimes*

Description

Prepare the point sets and bootstrap confidence curves used for the Supplementary Figure 4A-style rate-vs-variance and rate-vs-integration panels. Use `plot()` on the returned object to draw the panels.

Usage

```
regime_integration_relationships(
  summaries,
  resid_sd_threshold_vars = 2,
  resid_sd_threshold_corrs = 2,
  n_boot = 1000,
  ci_level = 0.99,
  seed = NULL,
  min_tips = NULL
)
```

Arguments

`summaries`	A list of manuscript-compatible `vars_cors` tables or a data frame from `summarize_regime_covariances` Regime IDs within each input table must be unique and non-empty. List inputs are pooled with a `run` column; missing or blank list names become `run1`, `run2`, and so on based on their positions, and the normalized names must be unique.
`resid_sd_threshold_vars`, `resid_sd_threshold_corrs`	Non-negative finite studentized-residual thresholds used to filter the variance and correlation panels.
`n_boot`	Number of bootstrap replicates for confidence curves.
`ci_level`	Confidence level for bootstrap ribbons.
`seed`	Optional random seed for reproducible bootstrap curves.
`min_tips`	Optional minimum tip count for inclusion when summaries contain a `tip_count` column. Relationship summaries are retained when `tip_count` >= `min_tips`; this differs from the strict PCA manuscript filter used by `regime_correlation_pca()`.

Details

High-correlation filtering is applied before this step with `summarize_regime_covariances()` via `remove_high_corr` and `corr_threshold`. The relationship object records residual filters and bootstrap settings, but it does not reapply matrix-level correlation filters. Rows with incomplete data for a panel remain in `combined` with an `NA` studentized residual and are omitted from that panel's point and removed-row data frames. Non-missing rates and variances must be finite and strictly positive, and non-missing correlations must be finite values in `[-1, 1]`. Boundary correlations at `-1` or `1` are

retained in combined, but their undefined Fisher-Z transforms and correlation-panel residuals are NA.

Value

An object of class regime_integration_relationships, containing plot-ready data frames, fitted lm objects, bootstrap curves, and settings.

Examples

```
summary_data <- data.frame(
  regime = paste0("r", 1:6),
  rate = c(1.0, 1.4, 1.7, 2.5, 3.1, 4.2),
  mean_variance = c(0.7, 0.9, 1.4, 1.8, 2.6, 3.4),
  mean_abs_correlation = c(0.12, 0.18, 0.23, 0.31, 0.37, 0.44)
)
relationships <- regime_integration_relationships(
  summary_data,
  n_boot = 20,
  seed = 1
)
stats::coef(relationships$variance_lm)
```

---

regime_module_diagnostics

Diagnose PCA Axes with User-defined Trait Modules

Description

Summarize within- and between-module correlations for each post-hoc regime and correlate those module summaries with regime correlation PCA scores.

Usage

```
regime_module_diagnostics(pca, modules, comparisons = NULL, pcs = NULL)
```

Arguments

pca	A regime_correlation_pca object from regime_correlation_pca() computed with use_correlation = TRUE.
modules	Named list of character vectors. Module names must be unique and not blank. Each element names the unique trait labels belonging to one anatomical, developmental, or functional module. Singleton modules are allowed; their within-module scores and PC correlations are NA because no pairwise correlation is defined.

comparisons	Optional named list defining module comparisons to score. Each element must be a length-two character vector naming entries in modules. When both names are the same, the score is the mean upper-triangle within-module correlation; otherwise it is the mean between-module correlation. If NULL, all within-module and pairwise between-module comparisons are generated. Missing or blank comparison names are generated from their module pairs; the resulting names must be unique and not blank.
pcs	Principal components to correlate with module summaries. Supply numeric indices or names such as "PC1". Defaults to all PCA score columns.

Value

An object of class regime_module_diagnostics, containing per-regime module scores, PC/module correlations, comparison definitions, and settings.

Examples

```
make_correlation <- function(r12, r13, r23) {
  matrix(
    c(1, r12, r13, r12, 1, r23, r13, r23, 1),
    nrow = 3,
    dimnames = list(c("bill", "wing", "tail"),
                    c("bill", "wing", "tail"))
  )
}
covariances <- list(
  r1 = make_correlation(0.1, 0.2, 0.3),
  r2 = make_correlation(0.2, 0.1, 0.4),
  r3 = make_correlation(-0.1, 0.3, 0.2),
  r4 = make_correlation(0.4, -0.2, 0.1)
)
pca <- regime_correlation_pca(covariances)
modules <- list(flight = c("wing", "tail"), feeding = "bill")
diagnostics <- regime_module_diagnostics(pca, modules)
diagnostics$correlations
```

---

runFalsePositiveSimulationStudy

Run a False-Positive Simulation Study

Description

Repeatedly simulate null datasets from a createSimulationTemplate() object and analyze each replicate with searchOptimalConfiguration() to estimate the expected false-positive behavior of a candidate bifrost search setup.

Usage

```
runFalsePositiveSimulationStudy(
  template,
  n_replicates,
  tree_tip_count = NULL,
  simulation_options = list(),
  search_options = list(),
  num_cores = 1,
  seed = NULL
)
```

Arguments

`template`	A `bifrost_simulation_template` returned by `createSimulationTemplate()`.
`n_replicates`	Integer number of simulation replicates to run.
`tree_tip_count`	Optional integer tip count for random subtree sampling. If `NULL`, each replicate uses the full empirical tree.
`simulation_options`	Named list of additional arguments passed to `simulateNullDataset()`, including `simulation_generator` and `covariance_df`. The manuscript-compatible `"original"` generator is used by default; request `simulation_generator = "empirical"` for full-covariance draws. The `simulation_generator` entry must contain exactly one supported string. For replicated studies, `simulation_options$seed` is not allowed; use the wrapper-level `seed` argument instead.
`search_options`	Named list of arguments passed to `searchOptimalConfiguration()`. These override the default response-only intercept-only search settings assembled from `template`. If `formula` is omitted, the wrapper inherits `template$search_formula`, which is intercept-only in the empirically calibrated simulation workflow.
`num_cores`	Integer number of workers used across replicate analyses. When greater than one, outer replicate parallelism takes precedence and each inner search is forced to `search_options$num_cores = 1`.
`seed`	Optional integer random seed. When supplied, the wrapper restores the caller's previous RNG state before returning.

Details

Simulated datasets, search results, a per-replicate summary table, and a compact study summary are returned in a single object of class `bifrost_simulation_study`.

Regardless of the global calibration model used to build `template`, the downstream search is intentionally restricted to intercept-only formulas. This keeps the simulation study focused on branch-shift detection in the response block, matching the residual-calibration logic. The simulated `trait_data` supplied to each search result therefore contains only the regenerated response variables. Templates calibrated with covariates are supported; those covariates influence the fitted means and residual covariance used to generate each response block, but they are not re-fit within each simulated search replicate.

Failed search replicates are retained in the output, but their inferred-shift and false-positive metrics are recorded as NA and excluded from scientific summaries. Their error messages and candidate counts remain available for diagnosis, and completion and failure rates are reported separately. Reproducibility for replicated studies is controlled by the wrapper-level seed argument. A supplied seed deterministically creates separate per-replicate simulation and search streams, so serial and parallel runs use the same replicate-level random draws. Parallel execution also uses future.seed = TRUE; per-replicate simulator seeds are intentionally disallowed here. If no replicate yields an evaluable false-positive rate, the study-level mean and median false-positive summaries are returned as NA.

Value

A list of class bifrost_simulation_study containing the simulated datasets, raw search results, a per-replicate summary table, and a compact study-level summary for the null scenario.

See Also

simulateNullDataset(), runShiftRecoverySimulationStudy(), searchOptimalConfiguration()

Examples

```
set.seed(1)
tr <- ape::rtree(14)
X <- matrix(rnorm(14 * 2), ncol = 2)
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")

fp_study <- runFalsePositiveSimulationStudy(
  tmpl,
  n_replicates = 1,
  tree_tip_count = 12,
  search_options = list(
    formula = "trait_data ~ 1",
    min_descendant_tips = 2,
    shift_acceptance_threshold = 5,
    num_cores = 1,
    IC = "GIC",
    method = "LL",
    plot = FALSE,
    progress = FALSE,
    verbose = FALSE,
    store_model_fit_history = FALSE
  ),
  num_cores = 1,
  seed = 2
)

fp_study
```

runSearchTuningGrid *Run a Fixed-IC Search Tuning Grid*

Description

Run a small simulation-based tuning grid for one information-criterion workflow at a time. This helper evaluates combinations of `shift_acceptance_threshold` and `min_descendant_tips` under a fixed `IC` value by combining:

- a null false-positive study,
- a proportional shift-recovery study, and
- a non-proportional correlation robustness study.

The goal is to support practical tuning workflows such as "find the best search settings under GIC" or "find the best search settings under BIC" without treating `IC` as just another free parameter in one large optimization.

Usage

```
runSearchTuningGrid(
  template,
  IC = c("GIC", "BIC"),
  shift_acceptance_thresholds,
  min_descendant_tips_values,
  tree_tip_count = NULL,
  null_replicates = 50,
  recovery_replicates = 50,
  null_simulation_options = list(),
  proportional_simulation_options,
  correlation_simulation_options = NULL,
  base_search_options = list(),
  fuzzy_distance = 2,
  weighted = TRUE,
  num_cores = 1,
  seed = NULL,
  store_studies = FALSE
)
```

Arguments

`template`	A `bifrost_simulation_template` returned by `createSimulationTemplate()`.
`IC`	Character scalar, either `"GIC"` or `"BIC"`. The tuning grid is run for this IC family only.
`shift_acceptance_thresholds`	Finite, non-negative numeric vector of candidate `shift_acceptance_threshold` values to evaluate.

`min_descendant_tips_values`
: Integer vector of candidate `min_descendant_tips` values to evaluate.

`tree_tip_count`
: Optional integer tip count passed to the simulation study wrappers. If `NULL`, each study uses the full empirical tree.

`null_replicates`
: Integer number of null replicates run for each grid row.

`recovery_replicates`
: Integer number of shifted replicates run for each grid row in both the proportional and correlation scenarios.

`null_simulation_options`
: Named list forwarded to `runFalsePositiveSimulationStudy()`. The `"original"` manuscript generator is used when `simulation_generator` is omitted. Its `simulation_generator` entry must contain exactly one supported string.

`proportional_simulation_options`
: Named list forwarded to `runShiftRecoverySimulationStudy()` for the proportional generating-model scenario. Must include `num_shifts`, `min_shift_tips`, and `max_shift_tips`. Its `simulation_generator` entry must contain exactly one supported string.

`correlation_simulation_options`
: Optional named list forwarded to `runShiftRecoverySimulationStudy()` for the non-proportional robustness scenario. The argument name is retained for compatibility with the `scale_mode = "correlation"` label. Under the default `"original"` generator it controls the published transform; under `simulation_generator = "empirical"` it controls the integration-rate trade-off. Its `simulation_generator` entry must contain exactly one supported string. If `NULL`, the proportional options are reused and only `scale_mode` is changed to `"correlation"`.

`base_search_options`
: Named list of search options shared across all grid rows. The tuning helper overwrites `IC`, `shift_acceptance_threshold`, and `min_descendant_tips` for each row. Simulation-grid searches are intentionally intercept-only, so `formula` should remain `"trait_data ~ 1"` (or an equivalent intercept-only response formula). If omitted, the helper inherits `template$search_formula`. Fitting options `method` and `error` inherit the template's settings unless explicitly overridden here; for example, `error = FALSE` can be used for searches on simulated data even when the template was fitted with `error = TRUE`.

`fuzzy_distance`
: Integer node distance passed to `evaluateShiftRecovery()` inside the shifted-study wrappers.

`weighted`
: Logical; if `TRUE`, request weighted recovery summaries from `runShiftRecoverySimulationStudy()` and include weighted fuzzy F1 columns in the output table.

`num_cores`
: Integer number of workers used across grid settings. When `num_cores > 1`, `runSearchTuningGrid()` parallelizes over settings and forces the dependent study wrappers and search calls to run serially within each setting.

`seed`
: Optional integer seed used to derive shared per-scenario study seeds. All grid rows use paired simulated datasets. Matching seeds and simulation arguments also pair separate GIC and BIC calls. When supplied, the wrapper restores the caller's previous RNG state before returning.

`store_studies` Logical; if `TRUE`, retain the raw study objects for every grid row and scenario. If `FALSE`, return only the summary table and metadata.

Details

This function is intentionally conservative in scope. It does not try to optimize over `IC`; instead, it assumes that GIC and BIC should be tuned as separate workflows. In typical use, you call it twice, once with `IC = "GIC"` and once with `IC = "BIC"`, then select one recommended setting from each grid with `selectTunedSearchParameters()`.

Within each scenario, every setting is evaluated on the same simulated replicates: trees, traits, and planted shifts are paired across settings, while different replicates remain independent simulation draws. The study wrappers regenerate these datasets deterministically using shared seeds and the L'Ecuyer-CMRG generator, so serial and parallel settings use the same datasets. Simulation and search seeds are managed separately. Pairing also applies when `seed = NULL`, but a new set of study seeds is drawn on each call; supply an explicit seed to reproduce a grid or pair separate IC calls. Changing grid order or adding thresholds does not change the simulated datasets. Matching across calls requires the same template, simulation options, replicate counts, and software/RNG configuration.

The template may come from a richer global calibration model, but each grid row is still evaluated with an intercept-only shift search on the simulated response block. This follows the residual-calibration workflow used by the simulation-study wrappers. Covariates in the calibration model influence the fitted means and residual covariance used for simulation, but they are not re-fit inside each grid replicate.

The returned `summary_table` contains one row per grid combination. Null summaries emphasize false-positive behavior, while proportional and correlation-scenario summaries emphasize recovery, including fuzzy balanced accuracy. The corresponding output columns retain their `correlation_` prefixes for backward compatibility. The `null_seed`, `proportional_seed`, and `correlation_seed` columns record the shared study seeds, including when `seed = NULL`. These repeat across grid rows rather than identifying independent datasets for each setting. Candidate-set availability among completed searches is tracked via evaluable fractions so that overly strict `min_descendant_tips` settings can be screened out before choosing a final workflow. Completion and failure rates use all attempted replicates and are reported separately. Scientific means, proportions, and recovery summaries are computed among completed searches, so failed searches are excluded from those summaries.

Parallelism is owned by the top-level function that the user calls. In this helper, `num_cores` is interpreted at the setting level, so dependent study wrappers are always called with `num_cores = 1` and their embedded search calls are forced to `num_cores = 1` as well. This avoids nested parallelism while keeping the user-facing API simple.

Value

A list of class `bifrost_search_tuning_grid` with components:

`IC` The fixed IC family used across the grid.

`paired_settings` Always `TRUE`: datasets are paired across settings.

`study_seeds` Named integer vector of the shared null, proportional, and correlation study seeds.

`grid` The evaluated combinations of thresholds and minimum clade sizes.

`summary_table` A data frame with one row per setting, deterministic per-study seed columns, and summary metrics from the null, proportional, and correlation-scenario studies.

`simulation_generators` A named character vector recording the null, proportional, and non-proportional simulation generators.

`studies` Either `NULL` or a list of raw study objects for each grid row and scenario, depending on `store_studies`.

`base_search_options` The shared search options used before the tuned grid parameters were injected, including inherited `method` and `error` settings.

See Also

`runFalsePositiveSimulationStudy()`, `runShiftRecoverySimulationStudy()`, `selectTunedSearchParameters()`

Examples

```
set.seed(1)
tr <- ape::rtree(16)
X <- matrix(rnorm(16 * 2), ncol = 2)
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")

gic_grid <- runSearchTuningGrid(
  template = tmpl,
  IC = "GIC",
  shift_acceptance_thresholds = 5,
  min_descendant_tips_values = 2,
  tree_tip_count = 14,
  null_replicates = 1,
  recovery_replicates = 1,
  null_simulation_options = list(simulation_generator = "empirical"),
  proportional_simulation_options = list(
    num_shifts = 1,
    min_shift_tips = 2,
    max_shift_tips = 5,
    scale_factor_range = c(5, 8),
    exclude_range = c(5.5, 6),
    buffer = 0,
    simulation_generator = "empirical"
  ),
  base_search_options = list(
    formula = "trait_data ~ 1",
    method = "LL",
    plot = FALSE,
    progress = FALSE,
    verbose = FALSE,
    store_model_fit_history = FALSE
  ),
  weighted = FALSE,
  num_cores = 1,
  seed = 2,
  store_studies = FALSE
```

```
)

gic_grid$summary_table[, c(
  "shift_acceptance_threshold",
  "min_descendant_tips",
  "null_mean_false_positive_rate",
  "proportional_evaluable_fraction",
  "proportional_strict_recall",
  "proportional_fuzzy_balanced_accuracy"
)]
```

`runShiftRecoverySimulationStudy` *Run a Shift-Recovery Simulation Study*

Description

Repeatedly simulate known-shift datasets from a `createSimulationTemplate()` object and analyze each replicate with `searchOptimalConfiguration()` to estimate expected shift recovery performance. The default `"original"` generator reproduces the published simulation operations. The `"empirical"` generator remains available explicitly for full-covariance draws and an empirically calibrated integration-rate robustness scenario.

Usage

```
runShiftRecoverySimulationStudy(
  template,
  n_replicates,
  tree_tip_count = NULL,
  simulation_options = list(),
  search_options = list(),
  fuzzy_distance = 2,
  weighted = TRUE,
  num_cores = 1,
  seed = NULL
)
```

Arguments

`template`	A `bifrost_simulation_template` returned by `createSimulationTemplate()`.
`n_replicates`	Integer number of simulation replicates to run.
`tree_tip_count`	Optional integer tip count for random subtree sampling. If `NULL`, each replicate uses the full empirical tree.

`simulation_options`
: Named list of additional arguments passed to `simulateShiftedDataset()`, including covariance and integration-rate trade-off controls. The manuscript-compatible `"original"` generator is used by default; request `simulation_generator = "empirical"` for the full-covariance option. The `simulation_generator` entry must contain exactly one supported string. For replicated studies, `simulation_options$seed` is not allowed; use the wrapper-level `seed` argument instead.

`search_options`
: Named list of arguments passed to `searchOptimalConfiguration()`. These override the default response-only intercept-only search settings assembled from `template`. If `formula` is omitted, the wrapper inherits `template$search_formula`, which is intercept-only in the empirically calibrated simulation workflow.

`fuzzy_distance`
: Integer node distance used for fuzzy matching in `evaluateShiftRecovery()`.

`weighted`
: Logical; if `TRUE`, compute weighted recovery summaries using IC weights when available.

`num_cores`
: Integer number of workers used across replicate analyses. When greater than one, outer replicate parallelism takes precedence and each inner search is forced to `search_options$num_cores = 1`.

`seed`
: Optional integer random seed. When supplied, the wrapper restores the caller's previous RNG state before returning.

Details

By default, the search configuration uses `uncertaintyweights_par = TRUE` when `weighted = TRUE`, so shift-recovery summaries can report both unweighted and IC-weighted metrics.

Regardless of the global calibration model used to build `template`, the downstream search is intentionally restricted to intercept-only formulas. This keeps the study focused on shift recovery in the simulated response block rather than re-estimating predictor effects within each replicate. The simulated `trait_data` supplied to each search result therefore contains only the regenerated response variables. Templates calibrated with covariates are supported; those covariates influence the fitted means and residual covariance used to generate each response block, but they are not re-fit within each simulated search replicate.

Failed search replicates are retained in the output, but their inferred-shift metrics are recorded as `NA` and excluded from recovery evaluation. Their error messages and candidate counts remain available for diagnosis, and completion and failure rates are reported separately. Reproducibility for replicated studies is controlled by the wrapper-level `seed` argument. A supplied seed deterministically creates separate per-replicate simulation and search streams, so serial and parallel runs use the same replicate-level random draws. Parallel execution also uses `future.seed = TRUE`; per-replicate simulator seeds are intentionally disallowed here. When `num_cores > 1`, namespace-level workers receive only their current replicate and compact shared settings; replicate call records use a literal template placeholder so the complete calibration template is not serialized again inside every simulated object.

Value

A list of class `bifrost_simulation_study` containing the simulated datasets, raw search results, per-replicate summaries, a study-level recovery summary, and the output of `evaluateShiftRecovery()`.

See Also

simulateShiftedDataset(), evaluateShiftRecovery(), runFalsePositiveSimulationStudy()

Examples

```
set.seed(1)
tr <- ape::rtree(16)
X <- matrix(rnorm(16 * 2), ncol = 2)
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")

recovery_study <- runShiftRecoverySimulationStudy(
  tmpl,
  n_replicates = 1,
  tree_tip_count = 14,
  simulation_options = list(
    num_shifts = 1,
    min_shift_tips = 2,
    max_shift_tips = 5,
    scale_mode = "proportional",
    scale_factor_range = c(5, 8),
    exclude_range = c(5.5, 6),
    buffer = 0
  ),
  search_options = list(
    formula = "trait_data ~ 1",
    min_descendant_tips = 2,
    shift_acceptance_threshold = 5,
    num_cores = 1,
    IC = "GIC",
    method = "LL",
    plot = FALSE,
    progress = FALSE,
    verbose = FALSE,
    store_model_fit_history = FALSE
  ),
  weighted = FALSE,
  num_cores = 1,
  seed = 2
)

recovery_study$per_replicate[
  , c("n_true_shifts", "n_inferred_shifts", "status")
]
unlist(recovery_study$evaluation$strict)
```

`searchOptimalConfiguration` *Search for a Multi-Regime Shift Configuration on a Phylogeny*

Description

Uses a greedy, single-pass search based on multivariate `mvgls()` fits. One-shift candidates are ranked once and then tested in that fixed order. The selected configuration is not guaranteed to be globally optimal. Optional per-shift IC weights compare the final model with refits that each omit one accepted shift.

The function fits the supplied response variables directly; it does not perform PCA. Models assume multi-rate Brownian motion with proportional VCV scaling across regimes.

Usage

```
searchOptimalConfiguration(
  baseline_tree,
  trait_data,
  formula = "trait_data ~ 1",
  min_descendant_tips = 10,
  num_cores = 2,
  ic_uncertainty_threshold = 1,
  shift_acceptance_threshold = 20,
  uncertaintyweights = FALSE,
  uncertaintyweights_par = FALSE,
  plot = FALSE,
  IC = "GIC",
  store_model_fit_history = TRUE,
  verbose = FALSE,
  ...,
  progress = TRUE,
  start_strategy = c("legacy_1.2.1", "native")
)
```

Arguments

`baseline_tree`	A rooted `phylo` (or SIMMAP/`phylo`) object representing the starting tree. It does not need to already be painted: existing SIMMAP states and within-edge segments are discarded before candidate generation, and the function internally paints a single baseline state at the root. Search candidates represent cladogenic shifts painted at descendant nodes. Tip labels must match `trait_data`.
`trait_data`	A `matrix` or `data.frame` of continuous trait values with row names matching `baseline_tree$tip.label` (same order). For the default `formula = "trait_data ~ 1"`, `trait_data` is typically supplied as a numeric matrix, but a numeric response-only `data.frame` is also accepted. When using more general formulas (e.g., pGLS-style models), a `data.frame` with named columns can be used instead.

`formula` Character string or formula object passed to `mvgls`. Defaults to `"trait_data ~ 1"`, which fits an intercept-only model treating the supplied multivariate trait matrix as the response. This is the appropriate choice for most morphometric data where there are no predictor variables. For more general models, `formula` can reference subsets of `trait_data` explicitly, for example `"trait_data[, 1:5] ~ 1"` to treat columns 1-5 as a multivariate response, `"trait_data[, 1:5] ~ trait_data[, 6]"` to fit a multivariate pGLS with an indexed predictor, or `cbind(y1, y2) ~ size + grp` to fit a named-column pGLS with numeric or factor predictors. Indexed predictor terms must each select one column: use `trait_data[, 3] + trait_data[, 4]` rather than `trait_data[, 3:4]`. Named formulas such as `cbind(y1, y2) ~ x1 + x2` are preferred for multiple predictors. An intercept is included by default; use `cbind(y1, y2) ~ 0 + size` or `cbind(y1, y2) ~ size - 1` to omit it. With a factor predictor, `cbind(y1, y2) ~ 0 + grp` estimates a separate mean for each group rather than fixing group means at zero.

`min_descendant_tips`
Integer ($\geq$2). Minimum number of tips required for an internal node to be considered as a candidate shift (forwarded to `generatePaintedTrees`). Defaults to `10`, the value evaluated and used in the focal analysis of Berv et al. (2026). Larger values reduce the number of candidate shifts by excluding very small clades. Smaller values trigger a runtime advisory and should be assessed for the dataset at hand. This filter is applied when the initial candidate pool is generated. It is not reapplied to each regime after shifts are accepted.

`num_cores` Integer. Maximum number of simultaneous fits during candidate scoring and parallel IC-weight estimation; defaults to `2`. The forward pass and serial IC weights run one fit at a time. See "Parallelization" for backend and numerical-thread behavior.

`ic_uncertainty_threshold`
Numeric ($\geq$0). Reserved for future development in post-search pruning and uncertainty analysis; currently not used by `searchOptimalConfiguration()`.

`shift_acceptance_threshold`
Numeric ($\geq$0). Minimum IC improvement (`current_best_ic - proposal_ic`) required to accept a candidate during the forward search. Larger thresholds require more improvement at each decision. Because the search is greedy, the final number and locations of shifts need not change monotonically with the threshold. Defaults to `20`, matching the focal ΔGIC setting of Berv et al. (2026); their GIC and BIC simulations also evaluated ΔIC = `10`. Values below `10` trigger a runtime advisory. These values are empirical reference points; users should explore alternative thresholds for their own datasets. The threshold must be a single finite nonnegative numeric value.

`uncertaintyweights`
Logical. If `TRUE`, compute per-shift IC weights serially by refitting the final model with each shift removed in turn. Exactly one of `uncertaintyweights` or `uncertaintyweights_par` must be `TRUE` to trigger IC-weight calculations; setting both to `TRUE` will result in an error. When enabled, the per-shift weights are returned in the `$ic_weights` component of the result.

`uncertaintyweights_par`
Logical. As above, but compute per-shift IC weights in parallel using **future.apply**. Exactly one of `uncertaintyweights` or `uncertaintyweights_par` must be

TRUE to trigger IC-weight calculations.

plot — Logical. If TRUE, draw/update a SIMMAP plot as the search proceeds (requires **phytools**).

IC — Character. Which information criterion to use, one of "GIC" or "BIC" (case-sensitive).

store_model_fit_history — Logical; TRUE by default. Retain each forward-proposal fit, its IC, acceptance decision, and any recorded error. Fits are written to temporary files during the search and loaded into the returned object afterward, which can require substantial storage and memory. Initial one-shift scoring fits and drop-one IC-weight refits are not included.

verbose — Logical. If TRUE, emit detailed candidate-generation, fitting, acceptance, rejection, and IC messages. By default, these details are emitted via message(). When plot = TRUE in an interactive RStudio session, they are written via cat() so they remain visible while plots are updating. This is independent of progress.

... — Additional arguments passed to mvgls (e.g., method, penalty, target, error, or REML). Setting error = TRUE asks mvgls() to estimate a nuisance measurement-error (intraspecific-variance) term. See the method guidance below.

progress — Logical. If TRUE (default), show three persistent CLI progress lines for candidate scoring, the greedy shift search, and optional IC-weight re-estimation. On dynamic terminals, the spinner redraws continuously while a model fit is running; counts, percentages, and ETA advance only after completed fits. To keep the main process responsive for these redraws, otherwise serial fits run one at a time in a background Future without imposing a one-thread numerical limit. Set to FALSE to disable these lines and the heartbeat path. Both settings preserve the same fitting random sequence for the same seed and search settings. Use progress = FALSE, verbose = FALSE to suppress progress and verbose messages; warnings and errors are still reported.

start_strategy — Starting-value policy for every baseline, candidate, forward-search, and IC-weight fit. The default, "legacy_1.2.1", computes data-dependent BM/BMM starts using the **mvMORPH** 1.2.1 development strategy used for the cached vignettes. This includes changes after the CRAN 1.2.1 release, particularly the random second-tip safeguard for singleton regimes. It passes these starting values to the installed mvgls(). Set "native" to use the installed **mvMORPH** initialization unchanged. Explicit start values or grid.search = FALSE in ... take precedence. EmpBayes requires "native". This compatibility policy preserves the older initialization, including its dependence on trait scale and random state. To reproduce an older sequential search, use the same seed, preceding searches, and serial/parallel settings. Turning progress on or off does not change the random sequence. This policy does not guarantee convergence or identical results across numerical environments or future **mvMORPH** versions.

Details

Input requirements.

- *Tree:* baseline_tree should be a rooted phylo tree with branch lengths expressed on a common scale. An ultrametric tree is not required. The starting tree does not need to already be

painted; searchOptimalConfiguration() paints a single baseline regime internally before building shifted candidates.

- *Trait data alignment:* rownames(trait_data) must match baseline_tree$tip.label in both names and order; any tips without data should be pruned beforehand.
- *Data type:* trait_data is typically a numeric matrix of continuous traits; numeric response-only data.frames are also supported for intercept-only searches, and named mixed-type data.frames are supported for richer formulas. High-dimensional settings ($p \geq n$) are supported via penalized-likelihood mvgls() fits.

Search outline. Fit the single-regime baseline and each eligible one-shift candidate. Rank candidates once by their IC improvement over the baseline. Visit that fixed list once, fitting each proposal on the current accepted tree. Accept it when its improvement over the current accepted model meets shift_acceptance_threshold. Rejected candidates are not revisited after later acceptances. Optional IC weights compare the final model with a refit omitting each accepted shift, using aicw.

If no eligible non-root candidates remain, or no proposed shifts are accepted, the returned final model is the single-regime baseline and optimal_ic equals baseline_ic.

Search-setting diagnostics. min_descendant_tips filters candidate nodes using their original number of descendant tips. It is applied only when the initial candidate pool is built; it is not reapplied to the resulting regimes after nested shifts are accepted. Consequently, a resulting regime can contain fewer tips than the threshold. An eligible small clade is evaluated like any other candidate and must pass shift_acceptance_threshold. Under Brownian motion, high trait disparity concentrated on short branches implies a high evolutionary-rate estimate, but a runtime rule cannot determine whether that pattern is biological or unstable. The permissive-settings advisory is therefore a guardrail rather than a power calculation. Examine per-shift IC weights by enabling uncertaintyweights or uncertaintyweights_par, repeat searches across plausible settings, and use dataset-specific simulations when conclusions depend on individual shifts.

Parallelization. num_cores caps simultaneous model fits: candidate scoring and parallel IC-weight re-estimation may use all requested workers, while the greedy search and serial IC weights always run one fit at a time. With progress enabled, fits run in Future workers so the main process can poll and redraw the spinner without advancing completion. Where forking is supported on Unix outside RStudio, multicore is used; otherwise multisession is used. Numerical-thread limits depend on the number of fitting workers, not on progress. With multiple workers, thread-limit environment variables and **RhpcBLASctl** runtime controls request one numerical thread in each worker. Single-worker stages preserve the caller's thread-limit environment, allowing within-fit multithreading when supported by the numerical library. For consistent behavior across fresh R sessions, configure numerical-thread environment variables before starting R; runtime-only changes in the main process are not automatically copied to multisession workers. Stage workers are shut down and the previous thread-limit environment and Future plan are restored afterward. Workers signal conditions through **progressr**; only the main R process renders progress, verbose messages, and plots.

Plotting. If plot = TRUE, trees are rendered with plotSimmap(); shift IDs are labeled with nodelabels().

Regime VCVs. For a multi-regime model, the returned $VCVs contain regime-specific covariance estimates extracted via extractRegimeVCVs. When no shifts are accepted, the final model is single-regime "BM": its trees retain baseline label "0", and VCVs[["0"]] contains the global covariance matrix from model_no_uncertainty$sigma$Pinv, including regularization when used. This entry does not represent an inferred shift or an additional fit; the model's param field remains NA.

Fitting methods. Choose an `mvgls` method appropriate for the formula and data. `"H&L"` with the `"RidgeArch"` penalty is restricted to intercept-only models. For models with predictors, `"LL"` is one option when an unregularized covariance can be estimated; `"PL-LOOCV"` is a penalized alternative. When `IC = "BIC"` and `method` is omitted, bifrost supplies `method = "LL"`. For `IC = "GIC"`, an omitted method is left to the installed **mvMORPH** default. See `mvgls` for method and penalty choices.

Value

A named `list` with (at minimum):

- `user_input`: captured call expressions, including resolved `progress` and `start_strategy` settings and flattened additional `mvgls()` arguments. This does not capture the seed, a data snapshot, or the full software environment.
- `initialization`: requested and actual starting policy for the baseline fit (`legacy_1.2.1`, `native`, `supplied`, or `native_no_grid`), plus the installed **mvMORPH** version. Each fitted model also records its own initialization information in its `bifrost_initialization` attribute.
- `tree_no_uncertainty_transformed`: SIMMAP tree from the final model on the transformed scale used internally by `mvgls`.
- `tree_no_uncertainty_untransformed`: same topology with original edge lengths restored.
- `model_no_uncertainty`: the final `mvgls` model object.
- `shift_nodes_no_uncertainty`: integer vector of accepted shift nodes; empty (possibly `NULL`) when no shifts are accepted.
- `optimal_ic`: final IC value; `baseline_ic`: baseline IC.
- `IC_used`: `"GIC"` or `"BIC"`; `num_candidates`: count of candidate one-shift models evaluated; `candidate_nodes`: integer node identifiers for those candidates.
- `model_fit_history`: if `store_model_fit_history = TRUE`, a list of per-iteration fits (loaded from temporary files written during the run) with proposal step, candidate node, regime ID, IC, status, acceptance decision, and any error, plus an `ic_acceptance_matrix` (IC value and acceptance flag per step).
- `VCVs`: named list of regime-specific VCV matrices extracted from the final model (penalized-likelihood estimates if PL was used). For a baseline-only BM fit, the sole entry `"0"` is its fitted global covariance matrix.

Additional components appear conditionally:

- `ic_weights`: a data frame with one row per accepted shift when `uncertaintyweights` or `uncertaintyweights_par` is `TRUE`. Each row compares the final model (`ic_with_shift`) with a refit omitting that shift (`ic_without_shift`). Here, `delta_ic = ic_with_shift - ic_without_shift`, so negative values favor retaining the shift. The two IC weights sum to one within each row; they are not normalized across shifts. `evidence_ratio` is `ic_weight_withshift / ic_weight_withoutshift`.
- `warnings`: when present, a list of warning and error messages collected during forward-proposal evaluation. It is not a complete record of warnings from every fitting stage or of optimizer convergence codes.

Convergence and robustness

The search is greedy and visits candidates in an initial ranked order. The selected configuration is not guaranteed to minimize the information criterion over all possible shift configurations.

Numerical fits can depend on starting values and optimizer stopping criteria with either `method = "H&L"` or `method = "LL"`. For conclusions that depend on individual shifts, repeat the search with the same data, tree, formula, and search settings using several recorded seeds when initialization is stochastic. Comparisons of supported `start_strategy` values can provide a separate initialization-sensitivity check. Record the software versions and parallel settings, and compare the accepted shifts, information-criterion values, and fitted rates.

Inspect optimizer codes and messages for the final model and saved forward proposals when available. A fit with a nonzero optimizer convergence code can still contribute an IC to candidate ranking or a forward acceptance decision. A nonzero code indicates that the optimizer reported a warning or failure; code 0 does not establish a global optimum. Inspect optimizer codes and messages rather than relying on `warnings` alone. Agreement across repeated runs is useful evidence of stability, but changing the seed may not change a deterministic initialization.

Compact example objects may omit the forward-proposal fits.

Separately, vary `shift_acceptance_threshold`, `min_descendant_tips`, or the information criterion to assess sensitivity to search settings. Per-shift IC weights are conditional on the selected fitted model; they do not assess convergence or alternative search paths.

References

Berv, J. S. et al. (2026). Rates of passerine body plan evolution in time and space. *Nature Ecology & Evolution*. doi:10.1038/s41559-026-03110-5.

See Also

`mvgls`, `GIC`, `BIC`, `icTrajectory` for extracting and plotting IC trajectories and shift acceptance decisions, and `generateViridisColorScale` for mapping regime-specific rates or parameters to a viridis color scale when plotting trees; packages: **mvMORPH**, **future**, **future.apply**, **phytools**, **ape**.

Examples

```
library(ape)
library(phytools)
library(mvMORPH)
set.seed(1)

# Simulate a tree
tr <- pbtree(n = 50, scale = 1)

# Define baseline regime "0" and high-rate regime "1" on a subset of tips
states <- setNames(rep("0", Ntip(tr)), tr$tip.label)
high_clade_tips <- tr$tip.label[1:20]
states[high_clade_tips] <- "1"

# Make a SIMMAP tree for the BMM simulation
simmap <- phytools::make.simmap(tr, states, model = "ER", nsim = 1)
```

```
# Simulate traits under a BMM model with ~10x higher rate in regime "1"
sigma <- list(
  "0" = diag(0.1, 2),
  "1" = diag(1.0, 2)
)
theta <- c(0, 0)

sim <- mvMORPH::mvSIM(
  tree  = simmap,
  nsim  = 1,
  model = "BMM",
  param = list(
    ntraits = 2,
    sigma   = sigma,
    theta   = theta
  )
)

# mvSIM returns either a matrix or a list of matrices depending on mvMORPH version
X <- if (is.list(sim)) sim[[1]] else sim
rownames(X) <- simmap$tip.label

# Run the search on the unpainted single-regime tree
res <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = X,
  formula                    = "trait_data ~ 1",
  min_descendant_tips        = 10,
  num_cores                  = 1,   # keep it simple / CRAN-safe
  shift_acceptance_threshold = 20,  # conservative GIC threshold
  IC                         = "GIC",
  plot                       = FALSE,
  store_model_fit_history    = TRUE,
  verbose                    = FALSE,
  progress                   = FALSE
)

res$shift_nodes_no_uncertainty
res$baseline_ic - res$optimal_ic  # total IC improvement
res$model_no_uncertainty$opt[c("convergence", "message")]
head(icTrajectory(res))           # forward proposals and decisions


# Intercept-only empirical-style search:
# H&L + measurement error
res_hl <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = X,
  formula                    = "trait_data ~ 1",
  min_descendant_tips        = 10,
  num_cores                  = 1,
  shift_acceptance_threshold = 20,
```

```
  IC                          = "GIC",
  plot                        = FALSE,
  method                      = "H&L",
  error                       = TRUE,
  store_model_fit_history     = FALSE,
  verbose                     = FALSE,
  progress                    = FALSE
)

# Formula-based search with a predictor:
# LL is one option for a model with predictors
dat <- data.frame(
  trait1    = X[, 1],
  trait2    = X[, 2],
  predictor = rnorm(nrow(X))
)
rownames(dat) <- simmap$tip.label

res_ll <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = dat,
  formula                    = "trait_data[, 1:2] ~ trait_data[, 3]",
  min_descendant_tips        = 10,
  num_cores                  = 1,
  shift_acceptance_threshold = 20,
  IC                         = "GIC",
  plot                       = FALSE,
  method                     = "LL",
  error                      = TRUE,
  store_model_fit_history    = FALSE,
  verbose                    = FALSE,
  progress                   = FALSE
)
```

`selectTunedSearchParameters` *Select Tuned Search Parameters from a Fixed-IC Grid*

Description

Choose a recommended `shift_acceptance_threshold` and `min_descendant_tips` combination from a `bifrost_search_tuning_grid` produced by `runSearchTuningGrid()`. Selection happens within a single IC family, making it easy to report one tuned workflow for GIC and another for BIC.

Usage

```
selectTunedSearchParameters(
  tuning_grid,
  max_false_positive_rate = 0.05,
```

```
  max_any_false_positive = 0.2,
  min_evaluable_fraction = 0.9,
 primary_metric = c("fuzzy_balanced_accuracy", "fuzzy_f1", "fuzzy_recall", "strict_f1",
    "weighted_fuzzy_f1"),
  scenario_weights = c(proportional = 0.5, correlation = 0.5),
  tie_break = c("conservative", "liberal"),
  allow_infeasible = FALSE
)
```

Arguments

`tuning_grid`	A `bifrost_search_tuning_grid` returned by `runSearchTuningGrid()`.
`max_false_positive_rate`	Maximum acceptable mean false-positive rate under the null study, as a finite number between zero and one.
`max_any_false_positive`	Maximum acceptable fraction of null replicates that infer at least one shift.
`min_evaluable_fraction`	Minimum acceptable candidate-set availability among completed searches. This filter is applied to the null, proportional, and integration-rate summaries; it is not a completion-rate threshold.
`primary_metric`	Character scalar indicating the recovery metric used to rank feasible settings. Defaults to `"fuzzy_balanced_accuracy"`. Supported legacy values are `"fuzzy_f1"`, `"fuzzy_recall"`, `"strict_f1"`, and `"weighted_fuzzy_f1"`.
`scenario_weights`	Numeric length-2 vector giving the weights applied to the proportional and integration-rate scenarios when forming the ranking score. The internal weight name `correlation` is retained for compatibility. If unnamed, the first value is used for proportional and the second for the integration-rate scenario.
`tie_break`	Character scalar. `"conservative"` prefers larger thresholds and larger `min_descendant_tips` when the primary ranking score is tied; `"liberal"` prefers the smaller values.
`allow_infeasible`	Logical. If `FALSE` (the default), stop without a recommendation when no rankable setting satisfies every constraint. Set to `TRUE` to retain the historical diagnostic fallback that ranks the full rankable grid and marks `used_all_settings = TRUE`.

Details

The selector uses a two-step decision rule. First, it filters out settings that violate the null false-positive or evaluability constraints. Second, it ranks the remaining settings using a weighted average of the chosen recovery metric across the proportional and integration-rate scenarios. By default, this ranking metric is fuzzy balanced accuracy.

Evaluable fractions measure candidate availability among completed searches. Completion and failure rates remain reported metadata based on all attempted replicates: the selector does not apply an automatic completion-rate filter.

Only settings with finite recovery metrics for every scenario having positive weight are rankable. If no rankable settings pass the filters, the function stops by default rather than returning an unsupported recommendation. With `allow_infeasible = TRUE`, it warns, falls back to the full rankable grid, and records that no feasible settings were available under the supplied constraints. If the grid contains no rankable settings, the function always stops.

Value

A list of class `bifrost_search_tuning_selection` with components:

`selected_row` The chosen row from `tuning_grid$summary_table`, augmented with a ranking score.

`recommended_search_options` A search-options list containing the tuned controls and fitting settings recorded by the simulation grid. For response-only empirical searches, the inherited intercept-only formula can usually be used directly. For empirical analyses with covariates, carry over the tuned controls but set `formula` to the empirical analysis formula. Other fitting options can also be explicitly changed for the empirical analysis, such as restoring `error = TRUE` after tuning with `error = FALSE`.

`feasible_table` The filtered and ranked table used for selection.

`n_feasible_settings` The number of settings that passed the supplied constraints.

`used_all_settings` Logical indicating whether the selector had to fall back to ranking the full grid because no feasible settings remained.

See Also

`runSearchTuningGrid()`, `searchOptimalConfiguration()`

Examples

```
summary_table <- data.frame(
  setting_id = 1:2,
  shift_acceptance_threshold = c(5, 10),
  min_descendant_tips = c(3L, 5L),
  null_mean_false_positive_rate = c(0.02, 0.01),
  null_fraction_any_false_positive = c(0.1, 0.1),
  null_evaluable_fraction = c(1, 1),
  proportional_evaluable_fraction = c(1, 1),
  correlation_evaluable_fraction = c(1, 1),
  proportional_fuzzy_balanced_accuracy = c(0.7, 0.8),
  correlation_fuzzy_balanced_accuracy = c(0.7, 0.8)
)
tuning_grid <- structure(
  list(
    IC = "GIC",
    summary_table = summary_table,
    base_search_options = list(formula = "trait_data ~ 1", method = "LL")
  ),
  class = c("bifrost_search_tuning_grid", "list")
)
```

```
tuned <- selectTunedSearchParameters(tuning_grid)
tuned$recommended_search_options
```

`shift_magnitude_counts` *Count Inferred Rate Increases And Decreases*

Description

Count inferred rate increases and decreases for a transition table, `bifrost_search` object, `shift_magnitude_comparison` object, or list of runs. This is the tabular input to `plot()` for shift-magnitude counts and reproduces the count/frequency summaries used by Berv et al. (2026) before plotting. Fitted `bifrost_search` inputs are accepted only for multi-regime BMM fits; generic mapped-tree inputs should use `tree` plus `state_values`.

Usage

```
shift_magnitude_counts(x = NULL, tree = NULL, state_values = NULL)
```

Arguments

`x`	A transition table, `bifrost_search` object, list of transition/search objects, `shift_magnitude_comparis` object, `shift_magnitude_groups` object, or `NULL` when using the `tree` argument.
`tree`	Optional SIMMAP-style `phylo` tree for generic input mode.
`state_values`	Named numeric state-value vector for generic input mode.

Value

A `shift_magnitude_counts` data frame with one row per input run. Columns include `source`, increase/decrease counts, total directional shifts, and increase/decrease frequencies. When `x` is a `shift_magnitude_groups` object, a named `shift_magnitude_count_set` is returned instead.

Examples

```
transitions <- data.frame(
  rate_change = c("root", "increase", "increase", "decrease", "decrease"),
  rate_delta = c(NA, 3, 4, -1, -2)
)
shift_magnitude_counts(list(run1 = transitions, run2 = transitions))
shift_magnitude_counts(shift_magnitude_groups(A = transitions, B = transitions))
```

shift_magnitude_groups *Define Shift-Magnitude Analysis Groups*

Description

Wrap named analysis groups so downstream shift-magnitude verbs know to run once per group rather than pooling every input into one result. Each group can be a transition table, BMM `bifrost_search` object, list of runs to pool, or a precomputed shift-magnitude object accepted by the downstream verb. When grouped inputs include precomputed `shift_magnitude_comparison` objects, `compare_shift_magnitudes()` requires their stored settings to match the settings requested for the grouped call.

Usage

```
shift_magnitude_groups(..., .list = NULL, labels = NULL)
```

Arguments

`...`	Named analysis groups. Use quoted names for labels that are not syntactic R names, such as `"GIC + BIC" = runs`.
`.list`	Optional named list of analysis groups. Use this when groups are already stored in a list.
`labels`	Optional replacement labels, one per analysis group. Defaults to names from the supplied groups, falling back to `"Analysis 1"`, `"Analysis 2"`, and so on when names are absent.

Value

A named list of class `shift_magnitude_groups`.

Examples

```
transitions <- data.frame(
  rate_change = c("increase", "increase", "decrease", "decrease"),
  rate_delta = c(2, 3, -1, -1.5)
)
groups <- shift_magnitude_groups(
  "GIC + BIC" = list(gic = transitions, bic = transitions),
  GIC = list(gic = transitions)
)
compare_shift_magnitudes(groups, ks_reps = 99)
```

shift_node_marks *Prepare And Plot Shift Node Marks On A Tree*

Description

shift_node_marks() prepares node markers for inferred regime shifts on an already plotted tree. For bifrost_search inputs, transitions are computed from the final multi-regime BMM mapped tree. Generic workflows can supply a shift_transitions() table, or a SIMMAP-style tree plus a named state_values vector.

Usage

```
shift_node_marks(
  x = NULL,
  tree = NULL,
  state_values = NULL,
  transitions = NULL,
  ic_weights = NULL,
  support_threshold = 0.9,
  letter_order = c("child_state", "node", "age"),
  marker_base_cex = 1.5,
  marker_scale_factor = 0.08,
  marker_transform = c("square_root", "log1p"),
  marker_max_cex = Inf
)

## S3 method for class 'shift_node_marks'
plot(
  x,
  rate_changes = c("increase", "decrease"),
  show_letters = TRUE,
  show_low_support = TRUE,
  show_legend = TRUE,
  marker_alpha = 0.75,
  letter_cex = 0.75,
  increase_fill = "white",
  decrease_fill = "#2b6cb0",
  marker_col = "#111827",
  low_support_cex = 0.7,
  low_support_col = grDevices::adjustcolor("black", alpha.f = 0.58),
  legend_position = "topleft",
  legend_cex = 0.72,
  ...
)

## S3 method for class 'shift_node_marks'
as.data.frame(
```

```
  x,
  row.names = NULL,
  optional = FALSE,
  component = c("marks", "increase_key", "summary"),
  ...
)
```

Arguments

x	A bifrost_search object, a shift_transitions() table, a SIMMAP-style phylo tree when state_values is supplied, or an object previously returned by shift_node_marks().
tree	Optional SIMMAP-style phylo tree for generic input mode.
state_values	Named numeric vector mapping SIMMAP state labels to state-associated values for generic input mode.
transitions	Optional precomputed transition table. When supplied, x is used only as a possible source of ic_weights.
ic_weights	Optional data frame with node and ic_weight_withshift columns. When omitted for a compatible bifrost_search object, x$ic_weights is used when available.
support_threshold	Numeric threshold used to flag low-support shifts from ic_weight_withshift.
letter_order	Ordering used when assigning letters to increase nodes. The default, "child_state", matches the Berv et al. (2026) manuscript convention by ordering increases by the destination regime state, with node number as a stable tie-breaker. Use "node" for node-number order or "age" for chronological order.
marker_base_cex, marker_scale_factor	Controls for marker size. Size is marker_base_cex + transformed(abs(percentage_change)) * marker_scale_factor.
marker_transform	Transformation applied to absolute percent change before scaling marker size.
marker_max_cex	Maximum marker size.
rate_changes	Which directional shifts to draw.
show_letters	Logical; draw letter labels for increase nodes.
show_low_support	Logical; draw an additional dot on shifts with ic_weight_withshift < support_threshold.
show_legend	Logical; draw a compact legend for the node marks.
marker_alpha	Alpha used for filled node markers.
letter_cex	Character expansion for letter labels.
increase_fill, decrease_fill	Fill colors for increase and decrease markers.
marker_col	Outline color for filled node markers.
low_support_cex, low_support_col	Size and color for low-support dots.

legend_position
: Position passed to graphics::legend().

legend_cex
: Character expansion passed to graphics::legend().

...
: Reserved for future extensions. Supplying unused arguments to plot() or as.data.frame() methods is an error.

row.names, optional
: Included for compatibility with base::as.data.frame(); ignored.

component
: Component to extract with base::as.data.frame(). Use "marks", "increase_key", or "summary".

Details

The workflow is intentionally plot-agnostic: first draw the tree with plot.rateMap() or another tree plotting function, then call plot() on the returned shift_node_marks object to add the node layer.

Value

shift_node_marks() returns an object of class "shift_node_marks" with marks, increase_key, low_support_nodes, summary, and settings. Plotting returns the same object invisibly. The as.data.frame() method returns the requested component as a plain data.frame.

Examples

```
toy_tree <- ape::read.tree(text = "(((a:1,b:1):1,c:2):1,d:3);")
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 3L,
  state = "2"
)
marks <- shift_node_marks(
  tree = toy_tree,
  state_values = c("0" = 1, "1" = 4, "2" = 2)
)
marks$increase_key
```

shift_transitions *Extract Regime Shift Transitions*

Description

Extract node-state transitions from a mapped SIMMAP tree and annotate them with state-associated numeric values. For `bifrost_search` objects, the values are the fitted BMM regime rates in `x$model_no_uncertainty$param`. `bifrost_search` inputs are accepted only for multi-regime BMM fits, because other model families do not expose the mapped heterogeneous-rate history summarized here. For generic inputs, supply a SIMMAP tree and a named numeric `state_values` vector.

Usage

```
shift_transitions(
  x = NULL,
  tree = NULL,
  state_values = NULL,
  include_root = TRUE
)
```

Arguments

`x`	A `bifrost_search` object, a compatible list with `tree_no_uncertainty_untransformed` and `model_no_uncertainty`, a SIMMAP-style `phylo` tree when `state_values` is supplied, or `NULL` when using the `tree` argument.
`tree`	Optional SIMMAP-style `phylo` tree for generic input mode.
`state_values`	Named numeric vector mapping SIMMAP state labels to state-associated values for generic input mode.
`include_root`	Logical; include a synthetic root row describing the root state and value.

Details

`shift_transitions()` implements the manuscript's node-based shift definition. Each positive-length edge map must therefore contain only one state. SIMMAP histories with state transitions inside an edge are rejected because assigning such a transition to the child node would give it the wrong time and branch location.

Value

A `shift_transitions` data frame with one row per detected parent-child node-state transition, plus the optional root row. Columns include node identity, node height and age, parent and child states, parent and child values, rate delta, percentage change, log rate ratio, and the classified `rate_change`. The result carries `tree`, `state_values`, and `settings` attributes that record the mapped tree, resolved state-value vector, and input options used to create the table.

Examples

```
toy_tree <- ape::read.tree(text = "(((a:1,b:1):1,c:2):1,d:3);")
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 3L,
  state = "2"
)
shift_transitions(
  tree = toy_tree,
  state_values = c("0" = 1, "1" = 4, "2" = 2)
)
```

---

shift_waiting_times *Compute Waiting Times Between Regime Shifts*

Description

Compute chronological whole-tree waiting times, between-shift intervals along root-to-tip lineages, and acceleration-focused increase-to-increase waiting times from a mapped SIMMAP tree or from the output of `shift_transitions()`. Lineage intervals exclude the censored root-to-first-shift and last-shift-to-tip boundaries. Fitted `bifrost_search` inputs are accepted only for multi-regime BMM fits; generic SIMMAP trees remain supported with `tree` and `state_values`. Tied or simultaneous transitions are preserved as zero-length waiting times in the descriptive tables.

Usage

```
shift_waiting_times(
  x = NULL,
  tree = NULL,
  state_values = NULL,
  scope = c("all", "global", "lineage", "acceleration")
)
```

Arguments

`x`	A BMM `bifrost_search` object, compatible list, SIMMAP tree, `shift_transitions` data frame, or `NULL` when using `tree`.

tree	Optional SIMMAP-style phylo tree for generic input mode, or the tree corresponding to a user-supplied transition table.
state_values	Named numeric state-value vector for generic input mode.
scope	Which components to compute. "all" computes global, lineage, and acceleration summaries.

Value

A list of class shift_waiting_times with transitions, global, lineage, lineage_by_tip, acceleration, and summaries components. Components outside the requested scope are returned as NULL.

Examples

```
toy_tree <- ape::read.tree(text = "(((a:1,b:1):1,c:2):1,d:3);")
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 1L,
  state = "0"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 2L,
  state = "1"
)
toy_tree <- phytools::paintSubTree(
  tree = toy_tree,
  node = ape::Ntip(toy_tree) + 3L,
  state = "2"
)
shift_waiting_times(
  tree = toy_tree,
  state_values = c("0" = 1, "1" = 4, "2" = 2)
)
```

---

simulateNullDataset *Simulate an Empirically Calibrated Null Replicate*

Description

Generate a single no-shift simulation replicate under a uniform Brownian motion process using the empirically calibrated covariance-generation framework. By default, each replicate uses the element-sampling operations from the published simulation code. The empirical generator is available explicitly for full-covariance draws centered on a createSimulationTemplate() object.

Usage

```
simulateNullDataset(
  template,
  tree_tip_count = NULL,
  seed = NULL,
  simulation_generator = c("original", "empirical"),
  covariance_df = NULL
)
```

Arguments

`template`	A `bifrost_simulation_template` returned by `createSimulationTemplate()`.
`tree_tip_count`	Optional integer tip count for random subtree sampling. If `NULL`, the full empirical tree from `template` is used.
`seed`	Optional integer random seed. When supplied, the function restores the caller's previous RNG state before returning.
`simulation_generator`	Simulation generator to use. `"original"`, the default, reproduces the element-sampling operations used in the published simulation code. `"empirical"` draws around the full fitted residual covariance. Supply exactly one supported string; explicitly supplied vectors are rejected. When omitted, `"original"` is used.
`covariance_df`	Optional integer degrees of freedom for empirical Wishart draws. It must be at least the number of response traits. The default uses the residual degrees of freedom stored in `template`.

Details

The default `"original"` generator samples marginal variance and covariance summaries and retains the published mathematical operations exactly. Under the explicit `"empirical"` generator, a new covariance matrix W is drawn as $W \sim Wishart(\nu, S/\nu)$, where S is the full empirical residual covariance and ν is `covariance_df`. Thus $E[W] = S$: empirical integration is retained on average while individual replicates vary.

For templates calibrated from richer global models, the empirical fitted mean structure is added back to the simulated residual process before the downstream intercept-only search is run on the regenerated response block. The returned `trait_data` therefore contains only the simulated response variables, even when the global calibration model included predictors. This is the lower-level replicate generator used by `runFalsePositiveSimulationStudy()`; use that wrapper for replicated false-positive simulation workflows.

Value

A list of class `bifrost_simulation_replicate_null` containing the sampled tree, a single-regime baseline tree, the simulated response matrix, a response-only `trait_data` object used for downstream model fitting, and the generated covariance matrix, generator metadata, resolved covariance degrees of freedom, and covariance diagnostics.

See Also

createSimulationTemplate(), simulateShiftedDataset(), runFalsePositiveSimulationStudy()

Examples

```
set.seed(1)
tr <- ape::rtree(12)
X <- matrix(rnorm(12 * 2), ncol = 2)
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")

sim_null <- simulateNullDataset(tmpl, tree_tip_count = 10, seed = 2)
dim(sim_null$trait_data)
```

---

simulateShiftedDataset

Simulate an Empirically Calibrated Shifted Replicate

Description

Generate a single known-shift simulation replicate using the empirically calibrated multi-shift framework. The function supports both the proportional generating-model scenario and a generator-specific non-proportional robustness scenario.

Usage

```
simulateShiftedDataset(
  template,
  tree_tip_count = NULL,
  num_shifts,
  min_shift_tips,
  max_shift_tips,
  scale_mode = c("proportional", "correlation"),
  scale_factor_range = c(0.1, 2),
  exclude_range = c(0.5, 1.5),
  buffer = 3,
  seed = NULL,
  simulation_generator = c("original", "empirical"),
  covariance_df = NULL,
  integration_power_range = c(0.5, 1.25),
  integration_exclude_range = c(0.8, 1.1),
  eigen_floor = 1e-08
)
```

Arguments

`template`	A `bifrost_simulation_template` returned by `createSimulationTemplate()`.
`tree_tip_count`	Optional integer tip count for random subtree sampling. If `NULL`, the full empirical tree from `template` is used.
`num_shifts`	Integer number of shifts to simulate.
`min_shift_tips`	Integer minimum size of any shifted clade.
`max_shift_tips`	Integer maximum size of any shifted clade.
`scale_mode`	Character string specifying the generating scenario: `"proportional"` for the manuscript generating model or `"correlation"` for the generator-specific non-proportional robustness scenario. With the default `"original"` generator, the latter reproduces the published correlation transform; with `simulation_generator = "empirical"`, it is an integration-rate trade-off.
`scale_factor_range`	Strictly positive numeric length-2 vector giving the full range of possible shift scalars.
`exclude_range`	Numeric length-2 vector specifying the central interval excluded from sampled shift scalars.
`buffer`	Integer minimum node distance between simulated shifts.
`seed`	Optional integer random seed. When supplied, the function restores the caller's previous RNG state before returning.
`simulation_generator`	Covariance and correlation-shift generator to use. `"original"`, the default, reproduces the published simulation operations exactly. `"empirical"` draws around the full fitted covariance and changes integration and marginal evolutionary variance in opposing directions. Supply exactly one supported string; explicitly supplied vectors are rejected. When omitted, `"original"` is used.
`covariance_df`	Optional integer degrees of freedom for empirical Wishart draws. It must be at least the number of response traits. The default uses the residual degrees of freedom stored in `template`.
`integration_power_range`	Strictly positive increasing numeric length-2 vector spanning one. Corrected correlation shifts draw spectral powers from the two tails of this range.
`integration_exclude_range`	Increasing numeric length-2 vector that contains one and lies strictly inside `integration_power_range`. Powers in this central interval are excluded so shifts have a material effect.
`eigen_floor`	Numeric stability floor, as a fraction of the largest eigenvalue, used by empirical correlation transforms.

Details

This function generates shifted residual processes around the fitted mean structure stored in the template:

- random subtree sampling from the empirical tree,

- shift placement subject to clade-size and non-overlap constraints,
- buffer enforcement using node distances,
- generation of a fresh ancestral covariance matrix centered on the full empirical residual covariance for each empirical replicate,
- construction of derived regimes by either proportional scaling or the generator-specific non-proportional robustness scenario, and
- simulation of multivariate BMM residuals that are added to the empirical fitted mean structure from the global calibration model.

Under `scale_mode = "proportional"`, derived regimes are scalar multiples of the ancestral covariance matrix and therefore match the generating assumptions of the current `bifrost` BMM search. Under `scale_mode = "correlation"`, the empirical generator raises the eigenvalues of the ancestral correlation matrix to a sampled positive power and renormalizes the result to a correlation matrix. Powers below one reduce integration; powers above one increase it. The derived covariance is then divided by the same power. Consequently, powers below one pair reduced integration with increased marginal evolutionary variance, whereas powers above one pair increased integration with decreased marginal variance. More precisely, `diag(Sigma_derived) = diag(Sigma_ancestral) / power`. This deliberate integration-rate trade-off remains a model-misspecification robustness test because the search fits proportional covariance shifts.

The `"original"` generator is supplied for exact reproduction. Its correlation scenario scales correlations and inversely scales marginal variances according to the original manuscript operations; it should not be interpreted as holding marginal variances fixed. Downstream simulation studies still evaluate intercept-only shift searches on the regenerated response block, so the returned `trait_data` contains only the simulated response variables. This is the lower-level shifted-replicate generator used by `runShiftRecoverySimulationStudy()` and `runSearchTuningGrid()`; use those wrappers for replicated shift-recovery or tuning-grid workflows.

Value

A list of class `bifrost_simulation_replicate_shifted` containing the painted generating tree, the true shift nodes, the simulated response matrix, a response-only `trait_data` object used for downstream model fitting, regime covariance matrices, sampled scale factors or integration powers, reciprocal marginal-variance scale factors, generator metadata, covariance diagnostics, and a scenario label. For schema compatibility, `sampledScaleFactors` contains the sampled integration powers under the empirical correlation generator; the explicit `sampledIntegrationPowers` and `sampledVarianceScaleFactors` fields should be preferred for interpretation.

See Also

`simulateNullDataset()`, `createSimulationTemplate()`, `runShiftRecoverySimulationStudy()`

Examples

```
set.seed(1)
tr <- ape::read.tree(
  text = "((((t1:1,t2:1):1,t3:2):1,t4:3):1,t5:4);"
)
X <- matrix(rnorm(5 * 2), ncol = 2)
```

```
rownames(X) <- tr$tip.label
tmpl <- createSimulationTemplate(tr, X, formula = "trait_data ~ 1", method = "LL")

sim_shift <- simulateShiftedDataset(
  tmpl,
  num_shifts = 1,
  min_shift_tips = 4,
  max_shift_tips = 4,
  scale_mode = "proportional",
  buffer = 0,
  seed = 3
)

sim_shift$shiftNodes
```

`summarize_regime_covariances` *Summarize Post-hoc Regime Covariance Matrices*

Description

Compute regime-rate, mean variance, mean absolute trait correlation, Fisher-Z transformed mean absolute correlation, tip count, and regime age from independent post-hoc covariance matrices.

Usage

```
summarize_regime_covariances(
  x,
  search = NULL,
  rates = NULL,
  tree = NULL,
  tip_counts = NULL,
  regime_ages = NULL,
  fisher_boundary = c("NA", "error"),
  remove_high_corr = FALSE,
  corr_threshold = 0.95
)
```

Arguments

`x`	A `regime_covariances` object returned by `fit_regime_covariances()` or a named list of covariance/correlation matrices.
`search`	Optional `bifrost_search` object used as a source of named regime-rate parameters and, when possible, mapped-regime tip counts/ages.
`rates`	Optional named numeric vector of regime rates. Names must be unique and non-empty and are matched to regime IDs; numeric equality is never used for matching.

`tree` Optional SIMMAP-style tree used to compute tip counts and regime ages when they are not already supplied.

`tip_counts` Optional named numeric vector of tip counts. When names are supplied, they must be unique and non-empty.

`regime_ages` Optional named numeric vector of regime ages. When names are supplied, they must be unique and non-empty.

`fisher_boundary`
How to handle correlations with `abs(r) >= 1`, where Fisher-Z is undefined. `"NA"` returns `NA` for those regimes; `"error"` stops with an error.

`remove_high_corr`
Logical; if `TRUE`, drop rows whose mean absolute correlation exceeds `corr_threshold`. This reproduces the manuscript `generateVarsCorsList(remove_high_corr = TRUE, corr_threshold = 0.95)` filtering step.

`corr_threshold` Correlation threshold used when `remove_high_corr` is `TRUE`.

Details

This summary is designed for independent post-hoc matrices. Do not pass proportional `search$VCVs` from a `bifrost_search` object when the question is whether regimes differ in phenotypic integration or correlation structure. If `x` is detectably the same object as `search$VCVs`, the function warns and still returns the requested descriptive summary. The single global covariance from an explicitly identified baseline-only BM fit is exempt from this warning; its scalar regime rate remains unavailable (`NA`). Raw matrix-list inputs must have unique, non-empty regime names. Their matrices must be symmetric and positive semidefinite, contain finite entries and strictly positive diagonal variances, and, when named, have unique matching row and column trait names; violations produce a regime-specific error. Symmetry and positive-semidefiniteness checks use a tolerance relative to the largest absolute matrix entry, independent of an overall change of units. Validation does not rescale the returned summaries. Invalid matrices encountered in a `regime_covariances` fit object are instead represented as failed rows with missing summaries and a diagnostic message. A one-trait matrix has no pairwise correlations, so its mean absolute correlation and Fisher-Z summary are returned as `NA`.

Value

A data frame with one row per regime and columns `regime`, `rate`, `mean_variance`, `mean_abs_correlation`, `fisher_z_mean_abs_correlation`, `tip_count`, `regime_age`, `status`, and `message`.

Examples

```
covariances <- list(
  slow = matrix(
    c(1, 0.3, 0.3, 2), nrow = 2,
    dimnames = list(c("bill", "wing"), c("bill", "wing"))
  ),
  fast = matrix(
    c(2, 0.8, 0.8, 3), nrow = 2,
    dimnames = list(c("bill", "wing"), c("bill", "wing"))
  )
)
```

```
summarize_regime_covariances(
  covariances,
  rates = c(slow = 0.8, fast = 1.6),
  tip_counts = c(slow = 12, fast = 9)
)
```

`summarize_regime_covariance_runs` *Summarize Post-hoc Regime Covariance Models Across Runs*

Description

Apply `summarize_regime_covariances()` to each element returned by `fit_regime_covariance_runs()`. This mirrors the manuscript `generateVarsCorsList()` step while keeping rate matching, tip counts, and high-correlation filtering explicit and reusable.

Usage

```
summarize_regime_covariance_runs(
  x,
  searches = NULL,
  rates = NULL,
  tree = NULL,
  tip_counts = NULL,
  regime_ages = NULL,
  fisher_boundary = c("NA", "error"),
  remove_high_corr = FALSE,
  corr_threshold = 0.95
)
```

Arguments

`x`	Named list of `regime_covariances` objects, usually from `fit_regime_covariance_runs()`. Missing or blank names are generated and names must be unique after normalization.
`searches`	Optional named list of corresponding `bifrost_search` objects. Run names are matched to `x`.
`rates`	Optional named list of regime-rate vectors, or one named rate vector to reuse for every run.
`tree, tip_counts, regime_ages`	Optional named lists of per-run values, or a single value to reuse for every run.
`fisher_boundary`	How to handle correlations with `abs(r) >= 1`, where Fisher-Z is undefined. `"NA"` returns `NA` for those regimes; `"error"` stops with an error.

`remove_high_corr`
: Logical; if `TRUE`, drop rows whose mean absolute correlation exceeds `corr_threshold`. This reproduces the manuscript `generateVarsCorsList(remove_high_corr = TRUE, corr_threshold = 0.95)` filtering step.

`corr_threshold`
: Correlation threshold used when `remove_high_corr` is `TRUE`.

Value

A named list of summary data frames with class `regime_covariance_run_summaries`.

Examples

```
slow <- matrix(c(1, 0.3, 0.3, 2), nrow = 2)
fast <- matrix(c(2, 0.8, 0.8, 3), nrow = 2)
covariance_runs <- list(
  first = list(slow = slow, fast = fast),
  second = list(slow = slow * 1.2, fast = fast * 0.9)
)
run_summaries <- summarize_regime_covariance_runs(
  covariance_runs,
  rates = list(
    first = c(slow = 0.8, fast = 1.6),
    second = c(slow = 0.9, fast = 1.4)
  )
)
run_summaries$first
```

Index

aicw, *85*
ape::all.equal.phylo(), *61*
ape::as.phylo(), *13*
as.data.frame.bifrost_search_tuning_grid, 4
as.data.frame.bifrost_search_tuning_selection, 4
as.data.frame.bifrost_shift_recovery_evaluation, 5
as.data.frame.bifrost_simulation_study, 6
as.data.frame.rate_distribution_fit *(*distribution-fit-data-frames*)*, 15
as.data.frame.regime_correlation_pca, 6
as.data.frame.regime_integration_relationship, 7
as.data.frame.regime_module_diagnostics, 8
as.data.frame.shift_magnitude_comparison *(*distribution-fit-data-frames*)*, 15
as.data.frame.shift_magnitude_comparison_set *(*distribution-fit-data-frames*)*, 15
as.data.frame.shift_magnitude_count_set *(*distribution-fit-data-frames*)*, 15
as.data.frame.shift_magnitude_counts *(*distribution-fit-data-frames*)*, 15
as.data.frame.shift_node_marks *(*shift_node_marks*)*, 94
as.data.frame.waiting_time_distribution_fit *(*distribution-fit-data-frames*)*, 15

base::as.data.frame(), *16*, *96*
BIC, *87*
bifrost_example_file, 9
bootstrap_rate_distribution, 9
bootstrap_rate_distribution(), *40*

compare_shift_magnitudes, 11
compare_shift_magnitudes(), *45*, *93*
createSimulationTemplate, 13
createSimulationTemplate(), *53*, *72, 73*, *75*, *79*, *99–103*

distribution-fit-data-frames, 15

evaluateShiftRecovery, 17
evaluateShiftRecovery(), *51*, *76*, *80, 81*

fisher_z_transform, 19
fit_rate_distribution, 20
fit_rate_distribution(), *10*, *40*
fit_regime_covariance_runs, 23
fit_regime_covariance_runs(), *106*
fit_regime_covariances, 21
fit_regime_covariances(), *23, 24*, *67*, *104*
fit_waiting_time_distribution, 24
future.apply::future_lapply(), *21*, *23*, *30*, *62*

generateViridisColorScale, 26, *87*
GIC, *87*
graphics::hist(), *41*
graphics::legend(), *35*, *96*
graphics::par(), *46*, *48*
graphics::plot(), *43*, *45, 46*, *48*
graphics::text(), *35*

icTrajectory, 27, *87*

lineage_rates, 29

mvgls, *82*, *84*, *86, 87*
mvMORPH::mvgls(), *13, 14*, *21–24*

nodelabels, *85*

phylolm::phylolm(), *69*
phytools::add.color.bar(), *37*
phytools::densityMap(), *39*, *55*, *59*
phytools::plotSimmap(), *37–39*, *55*, *59*
phytools::plotTree(), *37–39*
plot.icTrajectory, 34
plot.rate_distribution_fit, 40
plot.rateMap, 36
plot.rateMap(), *57*, *59*, *96*
plot.regime_correlation_pca, 41
plot.regime_integration_relationships, 42
plot.regime_module_diagnostics, 44
plot.shift_magnitude_comparison, 45
plot.shift_magnitude_comparison_set, 47
plot.shift_magnitude_count_set, 49
plot.shift_magnitude_counts, 47
plot.shift_node_marks *(*shift_node_marks*)*, 94
plotSimmap, *85*
print.bifrost_search, 49
print.bifrost_search_tuning_grid, 50
print.bifrost_search_tuning_selection, 50
print.bifrost_shift_recovery_evaluation, 51
print.bifrost_simulation_study, 51
print.bifrost_simulation_template, 52
print.rateMap, 53
progressr::with_progress(), *55*

rateMap, 54
rateMap(), *36*, *37*, *39*, *53*, *62*, *65*
rateMapControl, 61
rateMapControl(), *55–57*, *59*, *63*
rateMapRateFlags, 63
rateMapRateFlags(), *57*, *62*
rateMapView, 65
rateMapView(), *36*, *55*, *57*, *59*
regime_correlation_pca, 66
regime_correlation_pca(), *7*, *41*, *69–71*
regime_integration_pgls, 68
regime_integration_relationships, 70
regime_integration_relationships(), *42*
regime_module_diagnostics, 71
regime_module_diagnostics(), *44*
runFalsePositiveSimulationStudy, 72
runFalsePositiveSimulationStudy(), *14*, *52*, *76*, *78*, *81*, *100*, *101*
runSearchTuningGrid, 75
runSearchTuningGrid(), *50*, *51*, *76*, *89–91*, *103*
runShiftRecoverySimulationStudy, 79
runShiftRecoverySimulationStudy(), *14*, *17*, *18*, *52*, *74*, *76*, *78*, *103*

searchOptimalConfiguration, 82
searchOptimalConfiguration(), *14*, *17*, *27*, *49*, *72–74*, *79*, *80*, *91*
selectTunedSearchParameters, 89
selectTunedSearchParameters(), *50*, *51*, *77*, *78*
shift_magnitude_counts, 92
shift_magnitude_groups, 93
shift_magnitude_groups(), *12*
shift_node_marks, 94
shift_transitions, 97
shift_waiting_times, 98
shift_waiting_times(), *25*
simulateNullDataset, 99
simulateNullDataset(), *14*, *73*, *74*, *103*
simulateShiftedDataset, 101
simulateShiftedDataset(), *14*, *18*, *80*, *81*, *101*
stats::cov2cor(), *67*
stats::density(), *46*
stats::ks.test(), *12*
stats::prcomp(), *67*
summarize_regime_covariance_runs, 106
summarize_regime_covariances, 104
summarize_regime_covariances(), *68*, *70*, *106*

univariateML::bootstrapml(), *10*

viridis::viridis(), *26*