Package 'phylospatial'

November 17, 2026

Title Spatial Phylogenetic Analysis

Version 1.7.0

Description Analyze spatial phylogenetic diversity patterns.
Use your data on an evolutionary tree and geographic distributions of the terminal taxa to compute diversity and endemism metrics, test significance with null model randomization, analyze community turnover and biotic regionalization, and perform spatial conservation prioritizations. All functions support quantitative community data in addition to binary data. Methods are described in Kling (2025) <doi:10.1111/2041-210X.70056>.

License MIT + file LICENSE

Encoding UTF-8

Imports ape, sf, stats, terra, utils, vegan

Suggests canaper, furrr, future, testthat (>= 3.0.0), betapart, knitr, rmarkdown, hillR, picante, phytools, nullcat (>= 0.3.0), parallelDist, prioritizr (>= 9.0.0), highs, Matrix

URL https://matthewkling.github.io/phylospatial/, https://github.com/matthewkling/phylospatial

Config/testthat/edition 3

Depends R (>= 3.5)

VignetteBuilder knitr

BugReports https://github.com/matthewkling/phylospatial/issues

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.1

NeedsCompilation no

Author Matthew Kling [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9073-4240>)

Maintainer Matthew Kling <mattkling@berkeley.edu>

Repository CRAN

Date/Publication 2026-11-17 07:50:09 UTC

Contents

benefit . . . 2
clade_dist . . . 3
moss . . . 4
phylospatial . . . 5
plot.phylospatial . . . 8
plot_lambda . . . 9
ps_add_dissim . . . 10
ps_area . . . 10
ps_canape . . . 11
ps_canaper . . . 13
ps_dissim . . . 14
ps_dissim_focal . . . 16
ps_dissim_paired . . . 18
ps_diversity . . . 20
ps_expand . . . 22
ps_geodist . . . 23
ps_get_comm . . . 24
ps_grid . . . 25
ps_group_sizes . . . 27
ps_ordinate . . . 28
ps_performance . . . 29
ps_prioritize . . . 31
ps_prioritizr . . . 34
ps_quantize . . . 37
ps_rand . . . 38
ps_regions . . . 41
ps_regions_eval . . . 43
ps_rgb . . . 44
ps_simulate . . . 45
ps_suggest_n_iter . . . 46
ps_tile_groups . . . 47
ps_trace . . . 49
quantize . . . 50
to_spatial . . . 51
tree_scaling . . . 52

Index **55**

`benefit` *Calculate taxon conservation benefit*

Description

Nonlinear function that converts proportion of range conserved into conservation "benefit."

Usage

```
benefit(x, lambda = 1)
```

Arguments

x	Fraction of taxon range protected (value between 0 and 1).
lambda	Shape parameter.

Value

Value between 0 and 1.

---

clade_dist *Pairwise distances among clades or nodes*

Description

This function runs `ape::dist.nodes()` with some additional filtering and sorting. By default, it returns distances between every pair of non-nested clades, i.e. every pair of collateral (non-lineal) nodes including terminals and internal nodes. Package `phytools` is required for this function.

Usage

```
clade_dist(tree, lineal = FALSE, edges = TRUE)
```

Arguments

tree	A phylogeny of class `"phylo"`.
lineal	Logical indicating whether to retain distances for pairs of nodes that are lineal ancestors/descendants. If `FALSE` (the default), these are set to `NA`, retaining values only for node pairs that are collateral kin.
edges	Logical indicating whether to return a distance matrix with a row for every edge in `tree`. If `TRUE` (the default), rows/columns of the result correspond to `tree$edge`. If `FALSE`, rows/columns correspond to nodes as in `ape::dist.nodes()`.

Value

A matrix of pairwise distances between nodes.

Examples

```
if(requireNamespace("phytools", quietly = TRUE)){
  clade_dist(ape::rtree(10))
}
```

moss *Load California moss spatial phylogenetic data*

Description

Get example `phylospatial` data set based on a phylogeny and modeled distributions of 443 moss species across California. This data set is a coarser version of data from Kling et al. (2026). It contains occurrence probabilities, and is available in raster or polygon spatial formats. Companion layers giving existing land protection and human population density on the same spatial grid are also available, for use as `init` and `cost` inputs to conservation prioritization functions like `ps_prioritize()`.

Usage

```
moss(format = "raster", data = "phylospatial")
```

Arguments

`format`	Either "raster" (default) or "polygon"
`data`	Which data set to return. One of: • `"phylospatial"` (default): the moss spatial phylogenetic data set. • `"protection"`: the existing protection level of each grid cell, ranging from 0 (unprotected) to 1 (fully protected), based on the California Protected Areas Database (CPAD) and California Conservation Easement Database (CCED) as compiled by Kling et al. (2019). Values above 0.95 are set to 1. • `"popdens"`: mean human population density in each grid cell, in persons per square kilometer, from Gridded Population of the World v4 (2020). Because these values span several orders of magnitude and include zeros, they will usually need to be transformed before use as a prioritization `cost`.

Value

If `data = "phylospatial"`, a `phylospatial` object. Otherwise, a `SpatRaster` (if `format = "raster"`) or `sf` data frame (if `format = "polygon"`) with one variable, named after `data`, covering the same grid as the moss data set, with `NA` values for sites containing no moss taxa.

Source

Kling, Gonzalez-Ramirez, Carter, Borokini, and Mishler (2026) Spatial phylogenetics with continuous data: an application to California bryophytes. Systematic Botany 51(1): 101-120, https://doi.org/10.1600/036364426X17

Protection: Kling, Mishler, Thornhill, Baldwin, and Ackerly (2019) Facets of phylodiversity: evolutionary diversification, divergence and survival as conservation targets. Philosophical Transactions of the Royal Society B 374: 20170397, https://doi.org/10.1098/rstb.2017.0397.

Population density: Center for International Earth Science Information Network (CIESIN), Columbia University (2018). Gridded Population of the World, Version 4 (GPWv4): Population Density, Revision 11. NASA Socioeconomic Data and Applications Center (SEDAC), https://doi.org/10.7927/H49C6VHW.

Examples

```
moss()

# companion layers for conservation prioritization
protection <- moss(data = "protection")
popdens <- moss(data = "popdens")
terra::plot(c(protection, log1p(popdens)))
```

`phylospatial` *Create a spatial phylogenetic object*

Description

This function creates a `phylospatial` object. This is the core data type in the phylospatial library, and is a required input to most other functions in the package. The two essential components of a spatial phylogenetic object are a phylogenetic tree and an community data set.

Usage

```
phylospatial(
  comm,
  tree = NULL,
  spatial = NULL,
  data_type = c("auto", "probability", "binary", "abundance", "other"),
  clade_fun = NULL,
  build = TRUE,
  check = TRUE,
  area_tol = 0.01,
  rescale = c("sum1", "tip1", "raw"),
  site_weight = NULL
)
```

Arguments

`comm`	Community data representing the distribution of terminal taxa across sites. Can be a matrix with a column per terminal and a row per site, a SpatRaster with one layer per terminal, or a `sf` data with a column per terminal. Taxa whose names do not match between column/layer names in `comm` and tip labels in `tree` will be dropped with a warning (unless `build = FALSE`).
`tree`	Phylogeny of class phylo. Terminals whose names do not match `comm` will be dropped with a warning (unless `build = FALSE`). If this argument is not provided, terminals are assumed to follow a "star" tree with uniform branch lengths, which will lead to non-phylogenetic versions of any analyses done with the resulting `phylospatial` object. Must be a rooted tree.

spatial — An optional SpatRaster layer or sf object indicating site locations. The number of cells or rows must match comm. Ignored if comm is a SpatRaster or sf object.

data_type — Character giving the data type of comm. Must be "binary", "probability", "abundance", "auto" (the default), or "other". This determines how community values for clades are calculated from the values for terminal taxa. If "binary" (presence-absence), a clade is considered present in a site if any terminal in the clade is present. If "probability," clade probabilities are calculated as the probability that at least one terminal is present in a site. If "abundance," clade abundances are calculated as the sum of abundances for terminals in the clade in a site. If "auto," an attempt is made to guess which of these three data types was provided. This argument is ignored if clade_fun is provided, or if build = FALSE. If "other", a custom clade_fun must be supplied.

clade_fun — Function to calculate the local community weight for a clade based on community weights for tips found in a given location. Must be either NULL (the default, in which case the default function for the selected data_type is used) or a summary function that takes a numeric vector and returns a single numeric output. Ignored if comm already includes clade ranges.

build — Logical indicating whether comm already includes clade ranges that should be used instead of building new ones. Default is TRUE. If FALSE, clade_fun is ignored, no checks are performed to harmonize the tip labels and the community data, and the columns of comm must exactly match the order of tree edges including tips and larger clades. If clade ranges are included in comm but build = TRUE, they will be dropped and new clade ranges will be built.

check — Logical indicating whether community data should be validated. Default is TRUE.

area_tol — Numeric value giving tolerance for variation in the area of sites. Default is 0.01. If site_weight is NULL and the coefficient of variation in the area or length of occupied spatial units (e.g. grid cells) exceeds this value, a warning is issued, since sites will be weighted equally in calculating range sizes even though they differ in size. This argument is ignored if site_weight is provided, if check = FALSE, or if no spatial data is provided.

rescale — Character giving the branch-length rescaling method applied to the tree during construction. Must be "sum1" (default), "tip1", or "raw". "sum1" divides all branch lengths so they sum to 1. "tip1" divides all branch lengths by the longest root-to-tip path. "raw" applies no rescaling, preserving the original branch-length units. See rescale_tree() for details. Note that rescaling does not affect spatial patterns or statistical significance of diversity metrics—only their numeric scale.

site_weight — Optional weights determining how much each site contributes to the range sizes of the taxa that occur there. Can be NULL (the default), in which case all sites are weighted equally; "area", in which case sites are weighted by their area (or length, for lines) as calculated by ps_area(); or a numeric vector, single-layer SpatRaster, or sf object with one value per site (including unoccupied sites). Values must be positive for occupied sites, and are rescaled to have a mean of 1 across occupied sites. See the "Site weights" section below.

Details

This function formats the input data as a `phylospatial` object. Beyond validating, cleaning, and restructing the data, the main operation it performs is to compute community occurrence data for every internal clade on the tree.

Unoccupied sites (rows where no taxon occurs) are automatically removed from the community matrix during construction to improve performance. The original site indices of occupied rows are stored in `ps$occupied`, and the total number of sites (including unoccupied) in `ps$n_sites`, enabling reconstruction of full-extent spatial outputs. Functions that return spatial results automatically expand occupied-only data back to the full spatial extent.

If your data are in the form of occurrence point localities (e.g. from GBIF or BIEN) rather than a gridded community data set, use `ps_grid()` to rasterize the points onto a spatial grid before passing the result to this function.

Value

A `phylospatial` object, which is a list containing the following elements:

"data_type": Character indicating the community data type

"tree": Phylogeny of class `phylo`

"comm": Community matrix containing only occupied sites, including a column for every terminal taxon and every larger clade. Column order corresponds to tree edge order.

"spatial": A `SpatRaster` or `sf` providing spatial coordinates for all sites (including unoccupied). May be missing if no spatial data was supplied.

"occupied": Integer vector of row indices identifying which sites in the original data are occupied.

"n_sites": Total number of sites in the original data, including unoccupied.

"site_weight": Numeric vector of site weights for occupied sites, rescaled to a mean of 1, or `NULL` if all sites are weighted equally.

"dissim": A community dissimilarity matrix of class `dist` indicating pairwise phylogenetic dissimilarity between occupied sites. Missing unless `ps_add_dissim()` is called.

"rescale": Character indicating which branch-length rescaling was applied.

Site weights

A taxon's range size underlies endemism metrics and spatial prioritization: the endemism a taxon contributes to a site, and the value of protecting that site, both depend on the share of the taxon's range found there. By default, every site counts equally toward range size, so the range size of a taxon is the number of sites it occupies (or its total probability or abundance across sites). This is appropriate for equal-area grid cells, but not necessarily when sites differ in size, as with polygons of varying size or grid cells in a longitude/latitude coordinate system.

The `site_weight` argument controls how much each site counts. Weighting by area treats range size as the total area of the sites a taxon occupies. This is exact if community values represent the fraction of each site occupied by a taxon. But for presence data, it assumes that a taxon present in a site occupies all of it, overstating the range sizes of taxa found in large sites. Equal weights err in the opposite direction, counting presence in a large site no more than presence in a small one. Neither is correct in general. An intermediate option is to use weights of `ps_area(x)^z` with `0 < z < 1`,

reflecting the tendency of taxa to occupy a declining fraction of a site as site size increases. Because the appropriate value of z is generally unknown and varies among taxa, it can be useful to compare results across a range of values.

Site weights affect functions and metrics that depend on range size: the endemism metrics in `ps_diversity()`, `ps_rand()`, and `ps_canape()`; the `endemism` option of `ps_dissim()` and `ps_regions()`; and `ps_prioritize()`, `ps_prioritizr()`, and `ps_performance()`. They do not affect metrics that don't involve range size, such as PD. `ps_canaper()` does not support site weights.

See Also

`ps_grid()` to convert occurrence point data into a binary or abundance raster that can be used with phylospatial.

Examples

```
# load species distribution data and phylogeny
comm <- terra::rast(system.file("extdata", "moss_comm.tif", package = "phylospatial"))
tree <- ape::read.tree(system.file("extdata", "moss_tree.nex", package = "phylospatial"))

# construct `phylospatial` object
ps <- phylospatial(comm, tree)
ps
```

---

plot.phylospatial *Plot a* phylospatial *object*

Description

Plot a `phylospatial` object

Usage

```
## S3 method for class 'phylospatial'
plot(x, y = c("tree", "comm"), taxa = NULL, max_taxa = 12, ...)
```

Arguments

`x`	`phylospatial` object
`y`	Either `"tree"` or `"comm"`, indicating which component to plot.
`taxa`	Optional vector specifying which lineage ranges to plot if `y = "comm"`. Either a character vector of lineage names (column names of `x$comm`, i.e. tip labels for terminal taxa and `"clade1"`, `"clade2"`, etc. for larger clades) or an integer vector of column indices of `x$comm` (which correspond to the edges of `x$tree`). Ranges are plotted in the order given. If `NULL` (the default), a random sample of up to `max_taxa` lineages is plotted.

max_taxa	Integer giving the maximum number of randomly selected lineage ranges to plot if y = "comm" and taxa is NULL. Ignored if taxa is provided.
...	Additional arguments passed to plotting methods, depending on y and the class of x$spatial. For y = "tree", see plot.phylo; for y = "comm", see plot or plot.sf.

Value

A plot of the tree or community data.

Examples

```
ps <- ps_simulate(20, 20, 20)
plot(ps, "tree")
plot(ps, "comm")

# plot specific lineages, by name or by index
plot(ps, "comm", taxa = c("t1", "t2", "clade1"))
plot(ps, "comm", taxa = 1:4)
```

---

plot_lambda *Plot alternative lambda values*

Description

Show a plot illustrating alternative values for the lambda parameter in ps_prioritize. Lambda determines the shape of the "benefit" function that determines the conservation value of protecting a given proportion of the geographic range of a species or clade. Positive values place a higher priority on protecting additional populations of largely unprotected taxa, whereas negative values place a higher priority on protecting additional populations of relatively well-protected taxa. The default value used by ps_prioritize is 1.

Usage

```
plot_lambda(lambda = c(-1, -0.5, 0, 0.5, 2, 1))
```

Arguments

lambda	A vector of lambda values to plot

Value

Plots a figure

Examples

```
plot_lambda()
plot_lambda(seq(0, 3, .1))
```

ps_add_dissim *Add community dissimilarity data to a* phylospatial *object*

Description

This function calculates pairwise phylogenetic dissimilarity between communities and returns the phylospatial object with the dissimilarity data added as an element called dissim. See ps_dissim for details.

Usage

```
ps_add_dissim(ps, method = "sorensen", ...)
```

Arguments

ps	phylospatial data set.
method	Dissimilarity metric; see ps_dissim for details.
...	Additional arguments passed to ps_dissim, such as fun, endemism, or normalize.

Value

ps with a new dissim element added.

Examples

```
ps <- ps_simulate(data_type = "prob")
ps_add_dissim(ps)
ps_add_dissim(ps, fun = "vegdist", method = "jaccard", endemism = TRUE)
```

ps_area *Area of spatial units*

Description

Calculate the area (or length) of each site in a spatial data set or phylospatial object. The main use of this function is to derive site weights for phylospatial(); see the site_weight argument of that function for details.

Usage

```
ps_area(x)
```

Arguments

x	A phylospatial object, a SpatRaster, or an sf object. Must have a defined coordinate reference system.

Details

Raster cell areas are calculated with cellSize, which accounts for variation in cell area across latitudes in longitude/latitude rasters. Polygon areas and line lengths are calculated with expanse and perim, respectively. Point sites have no area, so every point is given a value of 1.

Value

A numeric vector with one value per site (including unoccupied sites, for a `phylospatial` object), giving area in square kilometers for raster cells and polygons, or length in kilometers for lines.

See Also

`phylospatial()`

Examples

```
# cells in a longitude/latitude raster shrink toward the poles
r <- terra::rast(nrows = 10, ncols = 10, xmin = -120, xmax = -100, ymin = 30, ymax = 60)
a <- ps_area(r)
range(a)

# one way to use these values as site weights:
comm <- terra::rast(r, nlyrs = 3)
names(comm) <- c("sp1", "sp2", "sp3")
set.seed(1)
comm[] <- rbinom(terra::ncell(comm) * 3, 1, .5)
tree <- ape::rtree(3, tip.label = names(comm))
ps <- phylospatial(comm, tree, site_weight = sqrt(a))
```

---

ps_canape *Categorical Analysis of Neo- and Paleo-Endemism (CANAPE)*

Description

This function classifies sites into areas of significant endemism according to the scheme of Mishler et al. (2014). Categorization is based on randomization quantile values for PE, RPE, and CE (which Mishler et al. call "PE on the comparison tree").

Usage

```
ps_canape(rand, alpha = 0.05)
```

Arguments

`rand`	An object returned by running `ps_rand`. It must include the metrics PE, RPE, and CE, and must have been computed with `summary = "quantile"`.
`alpha`	Numeric value between 0 and 1 giving the one-tailed p-value threshold to use when determining significance.

Details

Endemism significance categories are defined as follows:

- Endemism not significant: neither PE nor CE are significantly high at `alpha`.
- Significant neoendemism: PE or CE are significantly high at `alpha`; RPE significantly low at `alpha / 2` (two-tailed test).
- Significant paleoendemism: PE or CE are significantly high at `alpha`; RPE significantly high at `alpha / 2` (two-tailed test)..
- Significant mixed-endemism: PE or CE are significantly high at `alpha`; RPE not significant.
- Significant super-endemism: PE or CE are significantly high at `alpha / 5`; RPE not significant.

If the data have site weights, see the details of `ps_rand()` for how they affect the underlying null model tests.

Value

An object of the same class as `rand` containing a variable called `"canape"`, with values 0-4 corresponding to not-significant, mixed-, super-, neo-, and paleo-endemism, respectively. For a `SpatRaster`, the categories are labeled and the layer carries a color table with the conventional CANAPE colors (`"gray90"`, `"#CB7FFF"`, `"#9D00FF"`, `"red"`, and `"royalblue"`, in category order), which `terra::plot()` uses and `terra::writeRaster()` saves; pass `col` to `plot()` to use different colors. For other classes, `canape` is a factor with labeled levels.

References

Mishler, B. D., Knerr, N., González-Orozco, C. E., Thornhill, A. H., Laffan, S. W., & Miller, J. T. (2014). Phylogenetic measures of biodiversity and neo-and paleo-endemism in Australian Acacia. Nature Communications, 5(1), 4473.

Examples

```
# classic CANAPE using binary data and the curveball algorithm
# (note that a real analysis would require a much higher `n_rand`)
set.seed(123456)
ps <- ps_simulate(data_type = "binary")
rand <- ps_rand(ps, metric = c("PE", "RPE", "CE"),
                fun = "nullmodel", method = "curveball",
                n_rand = 25, burnin = 10000, progress = FALSE)
canape <- ps_canape(rand)
terra::plot(canape)
```

ps_canaper *Binary randomization tests including CANAPE*

Description

This function is a wrapper around `canaper::cpr_rand_test()`. It only works with binary community data. It is largely redundant with `ps_rand()` and `ps_canape()`, which are more flexible in supporting data sets with non-binary community data. However, this function runs faster, and supports custom null models via make.commsim.

Usage

```
ps_canaper(ps, null_model = "curveball", spatial = TRUE, ...)
```

Arguments

`ps`	phylospatial object
`null_model`	see `?canaper::cpr_rand_test()`
`spatial`	Logical: should the function return a spatial object (TRUE, default) or a vector (FALSE).
`...`	further arguments passed to `canaper::cpr_rand_test()`

Details

This function runs `canaper::cpr_rand_test()`; see the help for that function for details.

It also runs `canaper::cpr_classify_endem()` on the result, and includes the resulting classification as an additional variable, 'endem_type', in the output. 'endem_type' values 0-4 correspond to not-significant, neo, paleo, mixed, and super endemisim, respectively.

Value

A `matrix` or `SpatRaster`, or `sf` with a column or layer for each metric.

References

Mishler, B. D., Knerr, N., González-Orozco, C. E., Thornhill, A. H., Laffan, S. W., & Miller, J. T. (2014). Phylogenetic measures of biodiversity and neo-and paleo-endemism in Australian Acacia. Nature Communications, 5(1), 4473.

Nitta, J. H., Laffan, S. W., Mishler, B. D., & Iwasaki, W. (2023). canaper: categorical analysis of neo-and paleo-endemism in R. Ecography, 2023(9), e06638.

See Also

`ps_canape()`, `ps_rand()`

Examples

```
if(requireNamespace("canaper")){
      ps <- ps_simulate(data_type = "binary")
      terra::plot(ps_canaper(ps)$pd_obs_p_upper)
}
```

`ps_dissim` *Quantitative phylogenetic dissimilarity*

Description

This function calculates pairwise phylogenetic dissimilarity between communities. It works with both binary and quantitative community data sets. A wide range of phylogentic community dissimilarity metrics are supported, including phylogenetic Sorensen's and Jaccard's distances, turnover and nestedness components of Sorensen's distance (Baselga & Orme, 2012), and phylogenetic versions of all community distance indices provided through the vegan library. The function also includes options to scale the community matrix in order to focus the analysis on endemism and/or on proportional differences in community composition. The results from this function can be visualized using ps_rgb or ps_regions, or used in a variety of statistical analyses.

Usage

```
ps_dissim(
  ps,
  method = "sorensen",
  fun = c("vegdist", "designdist"),
  endemism = FALSE,
  normalize = FALSE,
  tips_only = FALSE,
  n_cores = NULL,
  ...
)
```

Arguments

`ps`	phylospatial object.
`method`	Character indicating the dissimilarity index to use: • "sorensen": Sorensen's dissimilarity, a.k.a. Bray-Curtis distance (the default) • "sorensen_turnover": The turnover component of Sorensen's dissimilarity, a.k.a. Simpson's. • "sorensen_nestedness": The nestedness component of Sorensen's dissimilarity.

- Any other valid `method` passed to `fun`. For options, see the documentation for those functions. The `vegdist` methods "morisita", "chao", and "cao" require integer counts, so they're only available for non-phylogenetic dissimilarity (`tips_only = TRUE`) of abundance data, without `endemism` or `normalize`.

`fun` Character indicating which general distance function from the `vegan` library to use: "vegdist" (the default) or "designdist". (While these functions are not explicitly designed to calculate phylogenetic beta diversity, their use here incorporates the phylogenetic components.) This argument is ignored if one of the three "sorensen" methods is selected.

`endemism` Logical indicating whether community values should be divided by taxon range sizes to derive endemism before computing distances. Range sizes are column totals, weighted by site weights if `ps` has them (see the `site_weight` argument of `phylospatial()`).

`normalize` Logical indicating whether community values should be divided by row totals (community sums) before computing distances. If `TRUE`, dissimilarity is based on proportional community composition. Normalization is applied after endemism.

`tips_only` Logical indicating whether to compute dissimilarity using only terminal taxa (`TRUE`) rather than the full phylogenetic community matrix (`FALSE`, the default). When `TRUE`, branch length weighting is skipped and the result is a standard (non-phylogenetic) community dissimilarity. Endemism and normalization options still apply.

`n_cores` Integer controlling the computation backend. The default `NULL` uses `parallelDist` with all available cores if installed, falling back to `vegan` otherwise. Setting `n_cores = 0` forces the `vegan` backend. Setting `n_cores` to a positive integer uses `parallelDist` with that many threads (requires `parallelDist` package). The `parallelDist` backend is faster than `vegan` even single-threaded for supported methods; unsupported methods (e.g. custom `designdist` formulas, turnover decomposition) always fall back to `vegan` with a message.

`...` Additional arguments passed to `fun`.

Value

A pairwise phylogenetic dissimilarity matrix of class `dist`.

References

Graham, C. H., & Fine, P. V. (2008). Phylogenetic beta diversity: linking ecological and evolutionary processes across space in time. Ecology Letters, 11(12), 1265-1277.

Baselga, A., & Orme, C. D. L. (2012). betapart: an R package for the study of beta diversity. Methods in Ecology and Evolution, 3(5), 808-812.

Pavoine, S. (2016). A guide through a family of phylogenetic dissimilarity measures among sites. Oikos, 125(12), 1719-1732.

See Also

ps_add_dissim(); ps_dissim_focal() for dissimilarity between focal sites and all other sites, including the asymmetric "focal_difference" index; and ps_dissim_paired() for dissimilarity between two data sets at each site, such as change over time.

Examples

```
# example data set:
ps <- ps_simulate(n_tips = 50)

# The default arguments give Sorensen's quantitative dissimilarity index
# (a.k.a. Bray-Curtis distance):
d <- ps_dissim(ps)

# Specifying a custom formula explicitly via `designdist`;
# (this is the Bray-Curtis formula, so it's equivalent to the prior example)
d <- ps_dissim(ps, method = "(b+c)/(2*a+b+c)",
      fun = "designdist", terms = "minimum", abcd = TRUE)

# Alternative arguments can specify a wide range of dissimilarity measures;
# here's endemism-weighted Jaccard's dissimilarity:
d <- ps_dissim(ps, method = "jaccard", endemism = TRUE)
```

---

ps_dissim_focal *Phylogenetic dissimilarity between focal sites and all other sites*

Description

Computes phylogenetic dissimilarity between one or more focal sites and every site in the data set, returning a map for each focal site. Mapping these values shows how lineage composition changes with distance from a focal location.

Usage

```
ps_dissim_focal(ps, site, method = "sorensen", spatial = TRUE, ...)
```

Arguments

ps	A phylospatial object.
site	The focal site(s), given as one of: • An integer vector of site indices, between 1 and ps$n_sites (cell numbers for raster data, or row numbers for polygon or matrix data). • A two-column matrix or data frame of x and y coordinates, in the coordinate reference system of ps$spatial. • An sf or SpatVector of points, which are reprojected to match ps$spatial if needed.

	Focal sites must be occupied.
method	Dissimilarity index: "focal_difference" (see Details), or any method accepted by ps_dissim().
spatial	Logical: if TRUE (the default) and ps has spatial data, return a SpatRaster or sf object; otherwise return a matrix with one row per site.
...	Further arguments passed to ps_dissim(), such as fun, endemism, normalize, or tips_only.

Details

This function shares its computations with ps_dissim(): for any symmetric index, ps_dissim_focal(ps, i, ...) gives the same values as the corresponding row of as.matrix(ps_dissim(ps, ...)). For the Sorensen family, "focal_difference", and custom formulas with fun = "designdist" (computed with vegan::designdist2()), values are computed for the focal sites only, without building the full pairwise matrix. Other vegdist methods compute the full matrix and return the relevant rows.

With fun = "designdist", the focal sites are the first data set, so in asymmetric formulas A (or a + b) refers to the focal site; for example, method = "(A-J)/A" with terms = "minimum" is equivalent to "focal_difference".

In addition to the methods available in ps_dissim(), this function offers method = "focal_difference", an asymmetric index: the fraction of the focal site's branch length that is absent from each other site. Writing a for the branch length shared by the focal site and another site, and b for the branch length found only at the focal site, it is $b/(a + b)$. It is 0 at the focal site, 1 at sites sharing none of the focal site's lineages, and, unlike symmetric indices, ignores lineages found only at the other site. It is the complement of a quantitative, phylogenetic dispersion field (Graves & Rahbek 2005). Because the comparison has a direction, it is only defined relative to a focal site and is not available in ps_dissim().

Value

A SpatRaster or sf object with one layer or variable per focal site, or a matrix with one column per focal site, giving values for every site (NA for unoccupied sites).

References

Graves, G. R., & Rahbek, C. (2005). Source pool geometry and the assembly of continental avifaunas. Proceedings of the National Academy of Sciences, 102(22), 7871-7876.

See Also

ps_dissim(), and ps_dissim_paired() for dissimilarity between two data sets at each site.

Examples

```
ps <- moss()

# dissimilarity from one site, given as a site (cell) index
site <- ps$occupied[250]
```

```
d <- ps_dissim_focal(ps, site)
terra::plot(d)

# turnover and nestedness components, from the same site
terra::plot(c(ps_dissim_focal(ps, site, method = "sorensen_turnover"),
              ps_dissim_focal(ps, site, method = "sorensen_nestedness")))

# fraction of the focal site's branch length absent from each site
terra::plot(ps_dissim_focal(ps, site, method = "focal_difference"))

# focal sites can also be given as coordinates
xy <- terra::xyFromCell(ps$spatial, ps$occupied[c(100, 400)])
terra::plot(ps_dissim_focal(ps, xy))
```

`ps_dissim_paired` *Phylogenetic dissimilarity between paired data sets*

Description

Computes phylogenetic dissimilarity between two data sets at each site, comparing every site in `x` to the same site in `y`. Typical uses include change over time (e.g. historical versus current distributions, or current versus projected), and comparisons of two estimates of the same communities (e.g. observed versus modeled).

Usage

```
ps_dissim_paired(
  x,
  y,
  method = "sorensen",
  tree = NULL,
  ranges = c("pooled", "x", "separate"),
  spatial = TRUE,
  ...
)
```

Arguments

`x, y`	`phylospatial` objects with the same sites.
`method`	Dissimilarity index: any method accepted by `ps_dissim()` except those noted in the description, or `"loss"` or `"gain"`.
`tree`	Optional phylogeny of class `phylo`, used to align the trees of `x` and `y` when they differ. See the description.
`ranges`	Range sizes used for endemism weighting: `"pooled"` (the default), `"x"`, or `"separate"`. Ignored unless `endemism = TRUE`. See the description.
`spatial`	Logical: if `TRUE` (the default) and `x` or `y` has spatial data, return a `SpatRaster` or `sf` object; otherwise return a matrix.
`...`	Further arguments: `fun`, `endemism`, `normalize`, and `tips_only`, as in `ps_dissim()`, and arguments passed to `fun`.

Details

Sites. `x` and `y` must have the same sites: the same number, and the same spatial data if both have it. Results cover every site occupied in either data set. A site occupied in only one data set has a Sorensen dissimilarity of 1; indices that are undefined when one community is empty, such as `"sorensen_turnover"`, or `"loss"` at a site empty in `x`, are `NA` there. Sites empty in both are `NA`.

Trees. By default, `x` and `y` must have identical trees. Because `phylospatial()` drops taxa with no occurrences, two data sets built from the same phylogeny will often have different trees, for example when a taxon occurs in only one time period. In that case, supply the original phylogeny as `tree`: the same tree used to build both data sets, including any transformation such as `delta_tree()`. It is pruned to the taxa found in either data set and rescaled with the data sets' `rescale` method, and clade ranges for both data sets are rebuilt on it from their terminal taxa. This gives the same result as if `phylospatial()` had kept the absent taxa. It requires that `x` and `y` were built from terminal taxa, rather than from supplied clade ranges (`build = FALSE`).

Methods. All methods available in `ps_dissim()` are supported, except the `vegdist` methods `"gower"`, `"mahalanobis"`, `"chisq"`, and `"raup"`, whose value for two communities depends on all the other sites in the data set. In addition, two asymmetric indices describe the direction of change. Writing a for the branch length shared by a site's two communities, b for the branch length found only in `x`, and c for the branch length found only in `y`:

- `"loss"`: $b/(a+b)$, the fraction of the branch length in `x` that is absent from `y`.
- `"gain"`: $c/(a+c)$, the fraction of the branch length in `y` that is absent from `x`.

As with the other quantitative indices, for probability or abundance data these include declines and increases in lineages present in both data sets, not only local losses and gains of lineages. With `fun = "designdist"`, `x` is the first data set, so in asymmetric formulas `A` (or `a + b`) refers to `x`.

Endemism. With `endemism = TRUE`, community values are divided by lineage range sizes before comparison. The `ranges` argument chooses which range sizes:

- `"pooled"` (the default): each lineage's combined range across both data sets. Both data sets are weighted on a common scale, and symmetric indices remain symmetric in `x` and `y`.
- `"x"`: ranges in `x`, treating it as a baseline. Lineages absent from `x` are weighted by their range in `y`. Symmetric indices are no longer symmetric in `x` and `y`.
- `"separate"`: each data set's own ranges, as in `ps_dissim()`. This conflates change in composition with change in range size: a lineage whose range contracts is weighted more heavily in `y`.

With `"pooled"` and `"x"`, `x` and `y` must have the same site weights (see the `site_weight` argument of `phylospatial()`).

Value

A `SpatRaster` with one layer or an `sf` object with one variable, or a one-column matrix with a row per site, giving dissimilarity at each site (`NA` for sites empty in both data sets).

See Also

`ps_dissim()`, `ps_dissim_focal()`

Examples

```
# a hypothetical second time period, in which some taxa are lost
# and others decline in occurrence probability
set.seed(1)
ps1 <- ps_simulate()
comm2 <- ps_get_comm(ps1)
i <- sample(terra::nlyr(comm2), 8)
comm2[[i[1:4]]] <- comm2[[i[1:4]]] * 0
comm2[[i[5:8]]] <- comm2[[i[5:8]]] * 0.5

# lost taxa are dropped from the second data set and its tree, with a warning
ps2 <- phylospatial(comm2, ps1$tree)

# so the original tree is supplied to align the two data sets
d <- ps_dissim_paired(ps1, ps2, tree = ps1$tree)
terra::plot(d)

# the direction of change
terra::plot(c(ps_dissim_paired(ps1, ps2, "loss", tree = ps1$tree),
              ps_dissim_paired(ps1, ps2, "gain", tree = ps1$tree)))
```

---

ps_diversity *Calculate spatial phylogenetic diversity and endemism metrics*

Description

This function calculates a variety of alpha phylogenetic diversity metrics, including measures of richness, regularity, and divergence. If continuous community data (probabilities or abundances) are provided, they are used in calculations, giving quantitative versions of the classic binary metrics.

Usage

```
ps_diversity(ps, metric = c("PD", "PE", "CE", "RPE"), spatial = TRUE)
```

Arguments

ps	phylospatial object (created by `phylospatial()` or `ps_simulate()`).
metric	Character vector containing the abbreviation for one or more diversity metrics listed in the details below. Can also specify `"all"` to calculate all available metrics. A small subset of common measures are selected by default.
spatial	Logical: should the function return a spatial object (TRUE, default) or a matrix (FALSE)?

Details

The function calculates the following metrics. Endemism-weighted versions of most metrics are available. All metrics are weighted by occurrence probability or abundance, if applicable.

Richness measures:

- **TD**—Terminal Diversity, i.e. richness of terminal taxa (in many cases these are species): $\sum_t p_t$
- **TE**—Terminal Endemism, i.e. total endemism-weighted diversity of terminal taxa, a.k.a. "weighted endemism": $\sum_t p_t r_t^{-1}$
- **CD**—Clade Diversity, i.e. richness of taxa at all levels (equivalent to PD on a cladogram): $\sum_b p_b$
- **CE**—Clade Endemism, i.e. total endemism-weighted diversity of taxa at all levels (equivalent to PE on a cladrogram): $\sum_b p_b r_b^{-1}$
- **PD**—Phylogenetic Diversity, i.e. total branch length occurring in a site: $\sum_b L_b p_b$
- **PE**—Phylogenetic Endemism, i.e. endemism-weighted PD: $\sum_b L_b p_b r_b^{-1}$
- **ShPD**—Shannon Phylogenetic Diversity, a.k.a. "phylogenetic entropy" (this version is the log of the "effective diversity" version based on Hill numbers): $-\sum_b L_b n_b log(n_b)$
- **ShPE**—Shannon phylogenetic Endemism, an endemism-weighted version of ShPD: $-\sum_b L_b n_b log(e_b) r_b^{-1}$
- **SiPD**—Simpson Phylogenetic Diversity: $1/\sum_b L_b n_b^2$
- **SiPE**—Simpson Phylogenetic Endemism, an endemism-weighted version of SiPD: $1/\sum_b L_b r_b^{-1} e_b^2$

Divergence measures:

- **RPD**—Relative Phylogenetic Diversity, i.e. mean branch segment length (equivalent to PD / CR): $\sum_b L_b p_b / \sum_b p_b$
- **RPE**—Relative Phylogenetic Endemism, i.e. mean endemism-weighted branch segment length (equivalent to PE / CE): $\sum_b L_b p_b r_b^{-1} / \sum_b p_b r_b^{-1}$
- **MPDT**—Mean Pairwise Distance between Terminals, i.e. the classic MPD metric. This is the average of cophenetic distances, weighted by p_t.
- **MPDN**—Mean Pairwise Distance between Nodes, an experimental version of MPD that considers distances between every pair of non-nested clades, putting more weight on deeper branches than does MPDT. This is the mean of distances between all collateral (non-lineal) node pairs including terminal and internal nodes, weighted by p_b.
- Note that divergence can also be assessed by using `ps_rand()` to run null model analyses of richness measures like PD.

Regularity measures:

- **VPDT**—Variance in Pairwise Distances between Terminals, i.e. the classic VPD metric, weighted by p_t.
- **VPDN**—Variance in Pairwise Distances between Nodes, i.e. MPDN but variance.

In the above equations, b indexes all taxa including terminals and larger clades; t indexes terminals only; p_i is the occurrence value (binary, probability, or abundance) of clade/terminal i in a given community; L_b is the length of the phylogenetic branch segment unique to clade b; and r_i is the range size of clade/terminal i, i.e. the sum of p_i across all sites, weighted by site weights if `ps` has them (see the `site_weight` argument of `phylospatial()`). For Shannon and Simpson indices, only nonzero elements of p_b are used, $n_b = p_b / \sum_b p_b L_b$, and $e_b = p_b / \sum_b p_b L_b r_b^{-1}$.

Sites of unequal size: Site weights affect only the endemism metrics, through range size. They are not applied to a site's own values, so like PD, endemism metrics describe the composition of a site rather than its size. To instead get each site's share of the tree's total branch length (the

classic interpretation of PE, which sums to the total length of the tree across sites), multiply PE by `ps$site_weight` (expanded to all sites with `ps_expand()`). Note that when sites differ in size, richness measures like TD and PD reflect site size as well as diversity, since larger sites tend to contain more taxa.

Value

A matrix, `sf` data frame, or `SpatRaster` with a column or layer for each requested diversity metric.

References

Faith, D. P. (1992). Conservation evaluation and phylogenetic diversity. Biological Conservation, 61(1), 1-10.

Laffan, S. W., & Crisp, M. D. (2003). Assessing endemism at multiple spatial scales, with an example from the Australian vascular flora. Journal of Biogeography, 30(4), 511-520.

Rosauer, D. A. N., Laffan, S. W., Crisp, M. D., Donnellan, S. C., & Cook, L. G. (2009). Phylogenetic endemism: a new approach for identifying geographical concentrations of evolutionary history. Molecular Ecology, 18(19), 4061-4072.

Allen, B., Kon, M., & Bar-Yam, Y. (2009). A new phylogenetic diversity measure generalizing the Shannon index and its application to phyllostomid bats. The American Naturalist, 174(2), 236-243.

Chao, A., Chiu, C. H., & Jost, L. (2010). Phylogenetic diversity measures based on Hill numbers. Philosophical Transactions of the Royal Society B: Biological Sciences, 365(1558), 3599-3609.

Mishler, B. D., Knerr, N., González-Orozco, C. E., Thornhill, A. H., Laffan, S. W., & Miller, J. T. (2014). Phylogenetic measures of biodiversity and neo-and paleo-endemism in Australian Acacia. Nature Communications, 5(1), 4473.

Tucker, C. M., Cadotte, M. W., Davies, T. J., et al. (2016) A guide to phylogenetic metrics for conservation, community ecology and macroecology. Biological Reviews, 92(2), 698-715.

Kling, M. M., Mishler, B. D., Thornhill, A. H., Baldwin, B. G., & Ackerly, D. D. (2019). Facets of phylodiversity: evolutionary diversification, divergence and survival as conservation targets. Philosophical Transactions of the Royal Society B, 374(1763), 20170397.

Examples

```
ps <- ps_simulate()
div <- ps_diversity(ps)
terra::plot(div)
```

`ps_expand` *Expand occupied-only results to full spatial extent*

Description

Takes a matrix or vector computed on occupied sites only and expands it back to the full site grid, inserting `NA` for unoccupied sites. This is useful when performing custom analyses on `ps$comm` (which contains only occupied sites) and mapping the results back to the full raster or spatial object.

Usage

```
ps_expand(ps, x, spatial = FALSE)
```

Arguments

ps	A phylospatial object.
x	A matrix (or vector) with nrow(ps$comm) rows (i.e. one row per occupied site).
spatial	Logical: if TRUE, convert the expanded result to a spatial object using ps$spatial. Default is FALSE.

Value

If spatial = FALSE, a matrix with ps$n_sites rows and NA for unoccupied sites. If spatial = TRUE, a SpatRaster or sf object.

Examples

```
ps <- ps_simulate()

# custom analysis on the occupied-only community matrix
site_totals <- matrix(rowSums(ps$comm), ncol = 1)
colnames(site_totals) <- "total"

# expand to full extent as a matrix
ps_expand(ps, site_totals)

# expand and convert to spatial
ps_expand(ps, site_totals, spatial = TRUE)
```

---

ps_geodist *Geographic distance between sites*

Description

Calculate pairwise geographic distances between occupied sites in a phylospatial object. The result is a dist object with the same dimensions and site ordering as ps_dissim(), making it straightforward to compare phylogenetic dissimilarity with geographic distance (e.g., for distance-decay analyses or Mantel tests).

Usage

```
ps_geodist(ps)
```

Arguments

ps	A phylospatial object with spatial data.

Details

For raster data, distances are computed between cell centroids. For polygon or line `sf` data, distances are computed between geometry centroids. Great-circle distances are used automatically when the data has a geographic (lon/lat) CRS; Euclidean distances are used for projected data or data without a CRS. Units are meters when a CRS is defined, or unitless coordinate distances otherwise.

Value

A pairwise geographic distance matrix of class `dist`, with one entry per pair of occupied sites. Units are meters when a CRS is defined, or raw coordinate units otherwise.

Examples

```
ps <- moss()
geo <- ps_geodist(ps)
phy <- ps_dissim(ps)

# distance-decay plot
plot(as.vector(geo), as.vector(phy),
     xlab = "Geographic distance (m)",
     ylab = "Phylogenetic dissimilarity",
     pch = ".", col = "#00000020")
```

---

ps_get_comm *Get* `phylospatial` *community data*

Description

Get `phylospatial` community data

Usage

```
ps_get_comm(ps, tips_only = TRUE, spatial = TRUE)
```

Arguments

`ps`	`phylospatial` object.
`tips_only`	Logical indicating whether only the terminal taxa (TRUE, the default) or all taxa (FALSE) should be returned.
`spatial`	Logical indicating whether a spatial (`SpatRaster` or `sf`) object should be returned. Default is `TRUE`; if `FALSE`, a matrix is returned.

Value

If spatial = TRUE, a SpatRaster or sf object with a layer/column for every taxon, with NA for unoccupied sites. If spatial = FALSE, a matrix containing only occupied sites (i.e., with nrow equal to the number of occupied sites, not the total number of grid cells). Use ps_expand() to expand an occupied-only matrix back to the full spatial extent if needed.

Examples

```
ps <- ps_simulate()

# the defaults return a spatial object of terminal taxa distributions:
ps_get_comm(ps)

# get distributions for all taxa, as a matrix
pcomm <- ps_get_comm(ps, tips_only = FALSE, spatial = FALSE)
```

---

ps_grid *Convert occurrence point data to a gridded community data set*

Description

This function takes a set of occurrence point localities (e.g. from GBIF or BIEN) and rasterizes them onto a spatial grid to produce a community data set suitable for passing to phylospatial(). Each species' point records are aggregated within grid cells to produce either binary presence-absence or count data.

Usage

```
ps_grid(
  x,
  grid = NULL,
  res = NULL,
  crs = NULL,
  cols = c(1, 2, 3),
  data_type = c("binary", "count")
)
```

Arguments

x	A data frame or sf points object containing occurrence records. If a data frame, it must contain columns for species identity and geographic coordinates (see cols). Coordinates in a data frame are assumed to be WGS84 longitude and latitude; an error is thrown if values fall outside valid ranges. If an sf object, it must contain a column for species identity and have point geometry; only the first element of cols is used.
grid	An optional SpatRaster to use as the target grid. If provided, res and crs are ignored. Point records falling outside the grid extent are silently dropped.

res	Numeric giving the grid cell resolution when generating a new grid. Units are meters if crs is provided or auto-generated, or in the units of crs otherwise. Ignored if grid is provided. If NULL (the default), a resolution is automatically chosen to produce approximately 1000 grid cells across the extent of the data. See Details.
crs	Optional CRS specification (any format accepted by terra::crs()) for the output grid. Ignored if grid is provided. If NULL (the default), an Albers Equal Area projection centered on the data is automatically generated. See Details.
cols	A vector of length 1 (for sf input) or 3 (for data frame input) identifying the columns in x for species, longitude, and latitude, in that order. Can be character column names or integer column indices but not a mix. Default is c(1, 2, 3). For sf input, only the first element (species column) is used; the remaining elements are ignored.
data_type	Character indicating the type of community data to produce. Either "binary" (default), in which cells receive a 1 if one or more records are present and 0 otherwise, or "count", in which cells receive the number of records.

Details

When grid is NULL, a grid is automatically generated from the point data. If crs is also NULL, an Albers Equal Area (AEA) projection is created based on the geographic extent of the points, with the central meridian and latitude at the midpoint of the coordinate ranges, and standard parallels at 1/6 and 5/6 of the latitude range. This ensures that grid cells are equal-area, so that every cell can count equally toward taxon range sizes (see the site_weight argument of phylospatial()). Users working with other spatial layers (e.g., environmental rasters) may want to supply a grid or crs that matches their existing data.

When res is NULL and no grid is supplied, the resolution is automatically chosen so that the grid contains approximately 1000 cells. Specifically, the resolution is set to sqrt(extent_area / 1000), where extent_area is the area of the bounding box of the projected points. The auto-generated grid is buffered by half a grid cell on each side to ensure that edge points are not excluded.

The res parameter is interpreted in the units of the output CRS. When the auto-generated AEA projection is used, units are meters. If a geographic (lon/lat) CRS is supplied, res is in degrees and the resulting cells will not be equal-area; a warning is issued in this case.

Coordinate cleaning and taxonomic name matching are outside the scope of this function. Users are encouraged to clean occurrence data beforehand (e.g., using the CoordinateCleaner package) and to ensure that species names in the occurrence data match the tip labels of their phylogeny before proceeding to phylospatial().

Value

A SpatRaster with one layer per species, suitable for passing directly to phylospatial() as the comm argument. Layer names correspond to unique values in the species column.

See Also

phylospatial() for constructing a spatial phylogenetic object from the output.

Examples

```
# simulate some occurrence records
set.seed(42)
occ <- data.frame(
  species = sample(paste0("sp", 1:10), 500, replace = TRUE),
  x = runif(500, -122, -118),
  y = runif(500, 34, 38)
)

# grid the occurrences with auto resolution (~1000 cells)
comm <- ps_grid(occ)
terra::plot(comm[[1:4]])

# specify columns by name
comm <- ps_grid(occ, cols = c("species", "x", "y"))

# grid at a specific resolution (50 km)
comm <- ps_grid(occ, res = 50000)

# use a custom grid
template <- terra::rast(res = 0.5, xmin = -123, xmax = -117, ymin = 33, ymax = 39,
                        crs = "EPSG:4326")
comm2 <- ps_grid(occ, grid = template)

# from sf points
occ_sf <- sf::st_as_sf(occ, coords = c("x", "y"), crs = 4326)
comm3 <- ps_grid(occ_sf, cols = "species")
```

---

ps_group_sizes *Average pool size of each site under a site grouping*

Description

For a grouping of sites, such as the tilings returned by `ps_tile_groups()` or a vector of fixed regions, computes the average number of occupied sites in each site's group (across columns, for a matrix of groupings). This is the size of the pool each site is randomized within when the grouping is passed to `ps_rand(site_groups = ...)`. Sites that are often in very small groups are rarely changed by fixed-margin randomizations; a site alone in its group is never changed.

Usage

```
ps_group_sizes(ps, groups, spatial = TRUE)
```

Arguments

`ps`	A `phylospatial` object.

groups	A grouping of sites in any form accepted by the site_groups argument of ps_rand().
spatial	Logical: return a spatial object (TRUE, the default) or a vector?

Value

A SpatRaster or sf object with a group_size variable, or a numeric vector with a value for every site (NA for unoccupied sites).

See Also

ps_tile_groups(), ps_rand()

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
  ps <- moss()
  tiles <- ps_tile_groups(ps, spacing = 200e3, n = 100)
  terra::plot(ps_group_sizes(ps, tiles))
}
```

---

ps_ordinate *Community phylogenetic ordination*

Description

Perform an ordination that reduces a spatial phylogenetic data set into k dimensions, using one of several alternative ordination algorithms.

Usage

```
ps_ordinate(ps, method = c("cmds", "nmds", "pca"), k = 3, spatial = TRUE)
```

Arguments

ps	A phylospatial object. Unless method = "pca", ps must have a non-null dissim component, generated by ps_add_dissim.
method	Ordination method, either "cmds" (the default, classical MDS, implemented via stats::cmdscale(), "nmds" (nonmetric MDS, implemented via vegan::metaMDS(); this is slower but often preferred), or "pca" (principal component analysis, implemented via stats::prcomp()),.
k	Positive integer giving the desired number of output dimensions; default is 3.
spatial	Logical indicating whether a spatial object (inherited from ps) should be returned. Default is TRUE.

Value

A matrix or spatial object with k variables.

See Also

For visualization using ordination onto RGB color space, see ps_rgb().

Examples

```
ps <- ps_add_dissim(ps_simulate(50, 5, 5))
ord <- ps_ordinate(ps, method = "cmds", k = 4)
terra::plot(ord)
```

---

ps_performance *Performance curves for a conservation prioritization*

Description

Compute performance curves describing how conservation value accumulates as sites are protected in the order given by a ps_prioritize() ranking.

Usage

```
ps_performance(ps, priority, target = 0.3)

## S3 method for class 'ps_performance'
plot(x, xvar = c("cost", "n_sites", "gain"), yvar = "value", ...)
```

Arguments

ps	The phylospatial object that was used to create priority.
priority	The result of ps_prioritize(). This must carry the "prioritization" attribute that ps_prioritize() attaches to its output; the attribute is lost if the result is written to file and read back in.
target	Numeric vector of one or more range protection targets, each between 0 and 1. For each target, the curve reports the fraction of the tree's total branch length belonging to lineages that have at least this fraction of their range protected. The default of 0.3 applies a 30% protection target to each lineage's range.
x	A ps_performance object.
xvar	Variable to plot on the x-axis: "cost" (the default), "n_sites", or "gain".
yvar	Variable to plot on the y-axis: "value" (the default) or one of the covX target columns.
...	Additional arguments passed to graphics::plot().

Details

Curves are reconstructed from the settings, inputs, and raw rankings stored with the prioritization result, so `init`, `cost`, `lambda`, and `protection` are those used in the original call to `ps_prioritize()`. The curve begins at step 0, which reflects any existing protection supplied via `init`. At each subsequent step, the next site in the ranking is raised to the `protection` level. Curves stop at the last ranked site, so they are truncated if `max_iter` was used. As in `ps_prioritize()`, the fraction of each lineage's range protected accounts for site weights if `ps` has them.

Network `value` is the conservation objective optimized by `ps_prioritize()`: the sum across all lineages of their `benefit()` (a function of the fraction of their range protected and the `lambda` parameter), weighted by their share of total tree length. It ranges from 0 (nothing protected) to 1 (every lineage's full range protected).

The function returns an error if `priority` lacks the prioritization attribute or if `ps` does not match the data set used for the prioritization, and gives a warning if the values in `priority` no longer match the stored rankings (e.g. because the object was modified after prioritization).

Value

A data frame of class `ps_performance`. For `method = "optimal"` prioritizations, and for `method = "probable"` prioritizations run with `summarize = FALSE`, it has one row per step per ranking, with columns:

ranking Index of the ranking (1 for an optimal prioritization; the rep number for probabilistic prioritizations).

step Step in the ranking, with 0 representing the starting state.

site Index of the site added at this step, in the full set of sites (including unoccupied sites).

site_cost, site_gain, site_value Cost of the site added at this step, the protection added (`protection` minus the site's `init` value), and the increase in network value.

n_sites, cost, gain, value Cumulative number of sites added, cost, protection added, and network value.

covX For each `target`, the fraction of total branch length belonging to lineages with at least X percent of their range protected.

For `method = "probable"` prioritizations run with `summarize = TRUE`, curves are summarized across reps at each step. The per-site columns are omitted, a `stat` column identifies the summary statistic (`"mean"` and percentiles `"pctX"`), and an `n_reps` column gives the number of reps that reached each step. Because these summaries are aligned by step, a summarized `cost` value represents the typical cost of reaching a given step, not a fixed budget.

See Also

`ps_prioritize()`

Examples

```
set.seed(123)
ps <- ps_simulate()
cost <- terra::setValues(ps$spatial, rep(seq(100, 20, length.out = 20), 20))
```

```
p <- ps_prioritize(ps, cost = cost, progress = FALSE)

perf <- ps_performance(ps, p, target = c(.1, .3))
head(perf)
plot(perf)
plot(perf, xvar = "n_sites", yvar = "cov30")

# probabilistic prioritization, with curves summarized across reps
pp <- ps_prioritize(ps, cost = cost, method = "prob",
      n_reps = 25, max_iter = 50, progress = FALSE)
plot(ps_performance(ps, pp))
```

`ps_prioritize` *Phylogenetic conservation prioritization*

Description

Create a ranking of conservation priorities using optimal or probabilistic forward stepwise selection. Prioritization accounts for the occurrence quantities for all lineages present in the site, including terminal taxa and larger clades; the evolutionary branch lengths of these lineages on the phylogeny, which represent their unique evolutionary heritage; the impact that protecting the site would have on these lineages' range-wide protection levels; the compositional complementarity between the site, other high-priority sites, and existing protected areas; the site's initial protection level; the relative cost of protecting the site; and a free parameter "lambda" determining the shape of the conservation benefit function.

Usage

```
ps_prioritize(
  ps,
  init = NULL,
  cost = NULL,
  lambda = 1,
  protection = 1,
  max_iter = NULL,
  method = c("optimal", "probable"),
  trans = function(x) replace(x, which(rank(-x) > 25), 0),
  n_reps = 100,
  n_cores = 1,
  summarize = TRUE,
  spatial = TRUE,
  progress = interactive()
)
```

Arguments

`ps`	phylospatial object.

`init` Optional numeric vector or spatial object giving the starting protection status of each site across the study area. Values should be between 0 and 1 and represent the existing level of conservation effectiveness in each site. If this argument is not specified, it is assumed that no existing reserves are present.

`cost` Optional numeric vector or spatial object giving the relative cost of protecting each site. Values should be positive, with greater values indicating higher cost of conserving a site. If this argument is not specified, cost is assumed to be uniform across sites.

`lambda` Shape parameter for taxon conservation benefit function. This can be any real number. Positive values, such as the default value 1, place higher priority on conserving the first part of the range of a given species or clade, while negative values (which are not typically used) place higher priority on fully protecting the most important taxa (those with small ranges and long branches) rather than partially protecting all taxa. See the function plot_lambda for an illustration of alternative `lambda` values.

`protection` Degree of protection of proposed new reserves (number between 0 and 1, with same meaning as `init`).

`max_iter` Integer giving max number of iterations to perform before stopping, i.e. max number of sites to rank.

`method` Procedure for selecting which site to add to the reserve network at each iteration:

- "optimal": The default, this selects the site with the highest marginal value at each iteration. This is a optimal approach that gives the same result each time.
- "probable": This option selects a site randomly, with selection probabilities calculated as a function of sites' marginal values. This approach gives a different prioritization ranking each time an optimization is performed, so `n_reps` optimizations are performed, and ranks for each site are summarized across repetitions.

`trans` A function that transforms marginal values into relative selection probabilities; only used if `method = "probable"`. The function should take a vector of positive numbers representing marginal values and return an equal-length vector of positive numbers representing a site's relative likelihood of being selected. The default function returns the marginal value if a site is in the top 25 highest-value sites, and zero otherwise.

`n_reps` Number of random repetitions to do; only used if `method = "probable"`. Depending on the data set, a large number of reps (more than the default of 100) may be needed in order to achieve a stable result. This may be a computational barrier for large data sets; multicore processing via `n_cores` can help.

`n_cores` Number of compute cores to use for parallel processing; only used if `method = "probable"`.

`summarize` Logical: should summary statistics across reps (TRUE, default) or the reps themselves (FALSE) be returned? Only relevant if `method = "probable"`.

`spatial` Logical: should the function return a spatial object (TRUE, default) or a matrix (FALSE)?

`progress` Logical: should a progress bar be displayed?

Details

This function uses the forward stepwise selection algorithm of Kling et al. (2019) to generate a ranked conservation prioritization. Prioritization begins with the starting protected lands network identified in `init`, if provided. At each iteration, the marginal conservation value of fully protecting each site is calculated, and a site is selected to be added to the reserve network. Selection can happen either in an "optimal" or "probable" fashion as described under the `method` argument. This process is repeated until all sites are fully protected or until `max_iter` has been reached, with sites selected early in the process considered higher conservation priorities.

A site's contribution to protecting a lineage depends on the share of the lineage's range found in the site. If `ps` has site weights (see the `site_weight` argument of `phylospatial()`), these shares account for them, so that sites with larger weights hold larger shares of the ranges of the taxa that occur there. Otherwise, all sites are weighted equally.

The benefit of the probabilistic approach is that it relaxes the potentially unrealistic assumption that protected land will actually be added in the optimal order. Since the algorithm avoids compositional redundancy between high-priority sites, the optimal approach will never place high priority on a site that has high marginal value but is redundant with a slightly higher-value site, whereas the probabilistic approach will select them at similar frequencies (though never in the same randomized run).

Every time a new site is protected as the algorithm progresses, it changes the marginal conservation value of the other sites. Marginal value is the increase in conservation benefit that would arise from fully protecting a given site, divided by the cost of protecting the site. This is calculated as a function of the site's current protection level, the quantitative presence probability or abundance of all terminal taxa and larger clades present in the site, their evolutionary branch lengths on the phylogeny, the impact that protecting the site would have on their range-wide protection levels, and the free parameter `lambda`. `lambda` determines the relative importance of protecting a small portion of every taxon's range, versus fully protecting the ranges of more valuable taxa (those with longer evolutionary branches and smaller geographic ranges).

Value

Matrix or spatial object containing a ranking of conservation priorities. Lower rank values represent higher conservation priorities. Sites that were never selected (because they ranked lower than `max_iter`, or because their `init` value was already at or above `protection`) have a rank value equal to the number of occupied sites in the data set (i.e. the lowest possible priority). Unoccupied sites are `NA`.

If `method = "optimal"`**.** the result contains a single variable "priority" containing the ranking.

If `method = "probable"` **and** `summarize = TRUE`**,** the "priority" variable gives the average rank across reps, variables labeled "pctX" give the Xth percentile of the rank distribution for each site, variables labeled "topX" give the proportion of reps in which a site was in the top X highest-priority sites, and variables labeled "treX" give a ratio representing "topX" relative to the null expectation of how often "topX" should occur by chance alone.

If `method = "probable"` **and** `summarize = FALSE`**,** the result contains the full set of `n_reps` solutions (variables "rep1", "rep2", etc.), each representing a ranking, with low values representing higher priorities.

The result also carries an attribute, "prioritization", recording the settings, inputs, and raw rankings used in the prioritization. This is used by ps_performance() to compute performance curves. The attribute is not preserved if the result is written to file.

References

Kling, M. M., Mishler, B. D., Thornhill, A. H., Baldwin, B. G., & Ackerly, D. D. (2019). Facets of phylodiversity: evolutionary diversification, divergence and survival as conservation targets. Philosophical Transactions of the Royal Society B, 374(1763), 20170397.

See Also

ps_performance() for performance curves; benefit() and plot_lambda() for the benefit function; ps_prioritizr() for conservation optimization with the prioritizr package.

Examples

```
# simulate a toy `phylospatial` data set
set.seed(123)
ps <- ps_simulate()

# basic prioritization
p <- ps_prioritize(ps)

# specifying locations of initial protected areas
# (can be binary, or can be continuous values between 0 and 1)
# here we'll create an `init` raster with arbitrary values ranging from 0-1,
# using the reference raster layer that's part of our `phylospatial` object
protected <- terra::setValues(ps$spatial, seq(0, 1, length.out = 400))
cost <- terra::setValues(ps$spatial, rep(seq(100, 20, length.out = 20), 20))
p <- ps_prioritize(ps, init = protected, cost = cost)

# using probabilistic prioritization
p <- ps_prioritize(ps, init = protected, cost = cost,
      method = "prob", n_reps = 1000, max_iter = 10)
terra::plot(p$top10)
```

---

ps_prioritizr *Build a prioritizr conservation problem*

Description

Convert a phylospatial object into a conservation planning problem for the prioritizr package, which finds optimal solutions using integer linear programming. Every branch of the phylogeny (terminal taxa and larger clades) is treated as a conservation feature, and range protection targets are applied to every branch. The function returns an unsolved problem with planning units, features, targets, and an objective; solvers, decision types, constraints, and penalties can then be added using prioritizr functions before solving the problem with prioritizr::solve(). See details for discussion of how this differs from prioritizr's own phylogenetic objective functions.

Usage

```
ps_prioritizr(
  ps,
  init = NULL,
  cost = NULL,
  protection = 1,
  objective = c("shortfall", "targets", "min_set"),
  target = 0.3,
  budget = NULL,
  spatial = TRUE
)
```

Arguments

ps	phylospatial object.
init, cost	Optional existing protection levels and protection costs for each site, as in ps_prioritize().
protection	Degree of protection of proposed new reserves (number between 0 and 1, with same meaning as init). Selecting a site in a solution raises its protection level to this value.
objective	Character indicating the optimization objective. All three objectives use the range protection target; see Details. • "shortfall" (the default): Minimize the branch-length-weighted relative shortfall from the target across all branches, subject to a budget. Partial progress toward the target counts, so this is a continuous version of "targets". Uses prioritizr::add_min_shortfall_objective(). • "targets": Maximize the fraction of total branch length belonging to branches that meet the target, subject to a budget. This optimizes the same quantity that ps_performance() reports in its covX columns. Uses prioritizr::add_max_n_targets_met_objective(). • "min_set": Find the lowest-cost set of sites that brings every branch up to the target. Uses prioritizr::add_min_set_objective().
target	Range protection target: a single number greater than 0 and no greater than protection, giving the fraction of each branch's range that should be protected (counting existing protection from init).
budget	Maximum total cost of newly selected sites. Required for the "shortfall" and "targets" objectives, and not used for "min_set".
spatial	Logical: should the problem use spatial planning units (TRUE, default)? If TRUE and ps contains spatial data, the problem is built on the SpatRaster or sf object in ps$spatial (with unoccupied sites excluded as planning units), and solutions are returned in the same format, with NA for unoccupied sites. This allows prioritizr's spatial penalties and constraints to calculate their own spatial data, but requires holding a spatial layer for every branch in memory. If FALSE, or if ps has no spatial data, the problem is built from a matrix of occupied sites, which is more memory-efficient; solutions are then returned as a numeric vector with an element for every occupied site, which can be mapped using ps_expand().

Details

Each site's contribution to a branch is the fraction of the branch's range that would be newly protected if the site were selected: the site's share of the branch's range (as in `ps_prioritize()`, based on occurrence probability or abundance for quantitative data, and accounting for site weights if `ps` has them), multiplied by `protection` minus the site's `init` value. Each branch's target is `target` minus its existing protection level, so that existing protection counts toward the target. For numerical stability, feature amounts are expressed as fractions of each branch's target (so that every target equals 1), and each branch's smallest amounts are dropped as long as their total is negligible (less than one millionth of the target), with the target reduced to match. Branches that already meet the target (to within one millionth of their range) are excluded from the problem; the names of the remaining features (`"edge1"`, `"edge2"`, etc.) refer to their column numbers in `ps$comm`.

For the `"shortfall"` and `"targets"` objectives, features are weighted by their branch lengths, with weights for `"shortfall"` adjusted for existing protection so that the objective is equivalent to maximizing the sum across branches of branch length times the fraction of the target achieved.

Sites whose `init` value is already at or above `protection` gain nothing from selection. They are locked into the solution with a cost of zero, so that spatial penalties (e.g. `prioritizr::add_boundary_penalties()`) treat existing reserves as part of the reserve network, and so that they do not count against the `budget`.

Note that prioritizr's evaluation functions, such as `prioritizr::eval_feature_representation_summary()`, report representation relative to the unprotected portion of each branch's range rather than its full range. Also note that prioritizr solvers default to a 10% optimality gap; set `gap = 0` when adding a solver if exact solutions are needed (e.g., when comparing solutions with `ps_prioritize()` results). Problems using the `"targets"` objective can take much longer to solve to optimality than the other objectives.

This approach differs from prioritizr's built-in phylogenetic objectives (`prioritizr::add_max_phylo_div_objective()` and `prioritizr::add_max_phylo_end_objective()`), which set targets for terminal taxa and credit a branch as conserved when at least one of its descendant taxa meets its target. Here, each clade's own range (the union of its descendants' ranges) is a feature with its own target, so deep branches count only once their own ranges are adequately protected. Compared with prioritizr's objectives that give special treatment to terminal taxa, this phylospatial version is more consistent with the clade-based definition of biodiversity that underpins Faith's PD and related metrics.

This function requires prioritizr version 9.0.0 or later, along with one of the solvers it supports.

Value

A `prioritizr::problem()` object.

See Also

`ps_prioritize()` for phylospatial's native stepwise prioritization algorithm.

Examples

```
if(requireNamespace("prioritizr", quietly = TRUE)){
      ps <- ps_simulate()

      # minimum-cost set of sites protecting 30% of every lineage's range
```

```
      prob <- ps_prioritizr(ps, objective = "min_set", target = .3)
      prob

      # maximize phylogenetic target coverage within a budget,
      # with existing protected areas and a boundary length penalty
      init <- terra::setValues(ps$spatial, rep(c(0, 1, 0, 0), each = 100))
      prob <- ps_prioritizr(ps, init = init, objective = "targets",
                            target = .5, budget = 50)
      prob <- prioritizr::add_boundary_penalties(prob, penalty = .01)

      if(requireNamespace("highs", quietly = TRUE)){
            sol <- solve(
                  prioritizr::add_highs_solver(prob, gap = 0, verbose = FALSE))
            terra::plot(sol)
      }
}
```

---

ps_quantize *Stratified randomization of a phylospatial object*

Description

Generates a randomized version of a phylospatial object by extracting the tip community matrix, permuting it using `nullcat::quantize()`, and rebuilding the phylospatial object using the permuted tip matrix.

Usage

```
ps_quantize(ps, site_groups = NULL, wt_row = NULL, wt_col = NULL, ...)
```

Arguments

`ps`	Object of class `phylospatial`
`site_groups`	Optional vector giving each site's group, so that sites are randomized independently within groups. One element per site (as for `site_weight` in `phylospatial()`) or per occupied site. See `ps_rand()` for details.
`wt_row`, `wt_col`	No longer supported, and must be `NULL` (the default); supplying a weight matrix gives an error. Use `site_groups` instead. See `ps_rand()` for details.
`...`	Additional arguments passed to quantize, such as `method`, `n_strata`, `transform`, `fixed`, `n_iter`, etc.

Details

The nullcat quantize routine involves three steps: converting a quantitative matrix to categorical strata, permuting the resulting categorical matrix using one of several categorical null model algorithms, and mapping the randomized categories back to quantitative values. Supply arguments via `...` to control options for each of these stages.

For repeated randomizations to generate a null distribution, it is more efficient to use `ps_rand(fun = "quantize")`, which is structured to avoid unnecessarily recomputing overhead that is shared across randomizations.

Value

A randomized version of ps

See Also

`ps_rand()`

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
  ps <- ps_simulate(data_type = "prob")
  ps_rand <- ps_quantize(ps, n_strata = 4,
    n_iter = 1000,
    method = "curvecat", fixed = "cell")
}
```

---

ps_rand *Null model randomization analysis of alpha diversity metrics*

Description

This function compares phylodiversity metrics calculated in ps_diversity to their null distributions computed by randomizing the community matrix or shuffling the tips of the phylogeny, indicating statistical significance under the assumptions of the null model. Various null model algorithms are available for binary, probability, and count data.

Usage

```
ps_rand(
  ps,
  metric = c("PD", "PE", "RPE", "CE"),
  fun = "tip_shuffle",
  method = NULL,
  n_iter = 1000,
  n_rand = 100,
  summary = "quantile",
  spatial = TRUE,
  n_cores = 1,
  progress = interactive(),
  site_groups = NULL,
  wt_row = NULL,
  wt_col = NULL,
  ...
)
```

Arguments

ps	phylospatial object.
metric	Character vector giving one or more diversity metrics to calculate; see ps_diversity for options. Can also specify "all" to calculate all available metrics.
fun	Null model function to use. Must be either "tip_shuffle", "nullmodel", "nullcat", "quantize", or an actual function: • "tip_shuffle" (the default): randomly shuffles the identities of terminal taxa. • "nullmodel": uses nullmodel and simulate.nullmodel, from the vegan package, which offer a wide range of randomization algorithms with different properties. • "nullcat": uses null model algorithms from the nullcat package. Only works with binary community data. This is the recommended path for binary data when mixing diagnostics (via ps_suggest_n_iter()) are desired. • "quantize": uses quantize, which converts a quantitative matrix to discrete strata, applies a categorical variant of the selected null model, and then maps randomized strata back to values. Only works with quantitative (probability or abundance) community data. • Any other function that accepts a community matrix as its first argument and returns a randomized version of the matrix.
method	Method used by the selected function. • For fun = "nullmodel", one of the method options listed under commsim. Be sure to select a method that is appropriate to your community data_type (binary, quantitative, abundance). • For fun = "nullcat" or fun = "quantize", one of the categorical algorithms listed under nullcat_methods (e.g. "curvecat", "swapcat"). • Ignored if fun is "tip_shuffle" or if it is a custom function.
n_iter	Integer giving the number of swap iterations per randomized matrix. Controls how thoroughly each null matrix is mixed before sampling. Default is 1000. Used with fun = "nullcat" (passed as n_iter to nullcat), fun = "quantize" (passed to quantize_prep), and fun = "nullmodel" with sequential methods like curveball (passed as burnin to simulate.nullmodel; ignored for non-sequential vegan methods). Use ps_suggest_n_iter() to estimate an appropriate value for your dataset.
n_rand	Integer giving the number of random communities to generate.
summary	Character vector of one or more summary statistics to return, each as a set of variables named with the statistic's first letter followed by the metric name (e.g. qPE, zPE, tPE): • "quantile" (the default): the proportion of randomizations in which the observed diversity metric was greater than the randomized metric, counting ties (randomizations equal to the observed value) as one half. A site whose randomized values all equal its observed value (for example, a site the null model cannot alter) therefore gets a quantile of 0.5. • "zscore": a "standardized effect size" or z-score relating the observed value to the mean and standard deviation of the randomizations.

- `"ties"`: the proportion `t` of randomizations whose value equals the observed value. Quantiles are bounded within `[t/2, 1 - t/2]`, so a site cannot be significant in either tail at level `alpha` when `t >= 2 * alpha`. Ties are common at sites a null model can rarely change, such as sites alone (or nearly so) in their group under `site_groups`.

Whatever the `summary`, a warning is given if the randomized values of some sites equal their observed values in every randomization.

`spatial` Logical: should the function return a spatial object (TRUE, default) or a matrix (FALSE).

`n_cores` Integer giving the number of compute cores to use for parallel processing.

`progress` Logical: should a progress bar be displayed?

`site_groups` Optional grouping of sites for a spatially constrained null model, used only with `fun = "nullcat"` or `fun = "quantize"`. Sites are randomized independently within each group, so taxa are only redistributed among sites in the same group. Either a vector giving each site's group (e.g. a region), applied to every randomization, or a matrix with one column per randomization (`n_rand` columns), giving a different grouping for each; `ps_tile_groups()` generates such a matrix of random spatial tilings, which pools each site with its neighbors. Rows (or elements) correspond either to all sites, as for the `site_weight` argument of `phylospatial()`, or to occupied sites only, as returned by `ps_tile_groups()`. With `site_groups`, `n_iter` iterations are run within every group; the number needed can be more or fewer than for the full matrix, so pass the same `site_groups` to `ps_suggest_n_iter()` to estimate it. See `nullcat::nullcat()` for details.

`wt_row, wt_col` No longer supported, and must be `NULL` (the default); supplying a weight matrix gives an error. Weighting which pairs of sites exchange values does not constrain the null model spatially; use `site_groups` instead. See `vignette("alpha-diversity")`.

`...` Additional arguments passed to the selected function: quantize, nullcat, simulate.nullmodel, or a custom function. Note that the `nsim` argument to simulate.nullmodel should not be used here; specify `n_rand` instead.

Details

If `ps` has site weights (see the `site_weight` argument of `phylospatial()`), they are used in calculating range sizes for endemism metrics, in both the observed and the randomized communities. Note, however, that null model algorithms that randomize the community matrix preserve taxon totals across sites, not weighted range sizes. With unequal site weights, a randomized taxon can therefore have a larger or smaller range size than the observed taxon, depending on which sites it is moved into, so null distributions of endemism metrics do not hold range sizes fixed. The default `"tip_shuffle"` method is not affected, since it moves intact terminal ranges among taxa.

Value

A matrix with a row for every row of `x`, a column for every metric specified in `metric`, and values for the `summary` statistic. Or if `spatial = TRUE`, a `sf` or `SpatRaster` object containing these data.

See Also

ps_diversity(), ps_tile_groups(), ps_canape()

Examples

```
# simulate a `phylospatial` data set and run randomization with default settings
ps <- ps_simulate(data_type = "prob")
rand <- ps_rand(ps)

# using the `quantize` function with the `curvecat` algorithm
if(requireNamespace("nullcat")){
    rand <- ps_rand(ps,
      fun = "quantize", method = "curvecat",
      transform = sqrt, n_strata = 4, fixed = "cell")
}

# binary data with nullcat's curvecat algorithm
ps2 <- ps_simulate(data_type = "binary")
if(requireNamespace("nullcat")){
    rand <- ps_rand(ps2, fun = "nullcat", method = "curvecat", n_iter = 1000)
}

# using binary data, with a vegan `nullmodel` algorithm
rand <- ps_rand(ps2, "PD", "nullmodel", "r2")

# using abundance data
ps3 <- ps_simulate(data_type = "abund")
rand <- ps_rand(ps3, metric = c("ShPD", "SiPD"),
      fun = "nullmodel", method = "abuswap_c")

# spatially constrained null model: randomize within a different random
# tiling (with 500 km spacing) in each randomization
if(requireNamespace("nullcat")){
    ps4 <- moss()
    tiles <- ps_tile_groups(ps4, spacing = 500e3, n = 20)
    rand <- ps_rand(ps4, metric = "PE", fun = "quantize", n_rand = 20,
                    zero_stratum = TRUE, n_iter = 5000,
                    site_groups = tiles, summary = c("quantile", "ties"))
}
```

---

ps_regions *Cluster analysis to identify phylogenetic regions*

Description

Perform a clustering analysis that categorizes sites into biogeographic regions based on phylogenetic community compositional similarity.

Usage

```
ps_regions(ps, k = 5, method = "average", endemism = FALSE, normalize = TRUE)
```

Arguments

ps	A phylospatial object. If method is anything other than "kmeans", it must contain a dissim component generated by ps_add_dissim.
k	Number of spatial clusters to divide the region into (positive integer). See ps_regions_eval to help choose a value of k by comparing the variance explained by different numbers of regions.
method	Clustering method. Options include all methods listed under hclust, and "kmeans". If "kmeans" is selected, the dissim component of ps is ignored.
endemism	Logical indicating whether community values should be divided by taxon range sizes (column totals, weighted by site weights if ps has them) to derive endemism. Only used if method = "kmeans"; in other cases this information should instead be supplied to ps_add_dissim.
normalize	Logical indicating whether community values should be divided by row totals (community sums). If TRUE, dissimilarity is based on proportional community composition. This happens after endemism is derived. Only used if method = "kmeans"; in other cases this information should instead be supplied to ps_add_dissim.

Value

A raster or matrix with an integer indicating which of the k regions each site belongs to.

References

Daru, B. H., Elliott, T. L., Park, D. S., & Davies, T. J. (2017). Understanding the processes underpinning patterns of phylogenetic regionalization. Trends in Ecology & Evolution, 32(11), 845-860.

Examples

```
ps <- ps_simulate()

# using kmeans clustering algorithm
terra::plot(ps_regions(ps, method = "kmeans"))

# to use a hierarchical clustering method, first we have to `ps_add_dissim()`
terra::plot(ps_regions(ps_add_dissim(ps), k = 7, method = "average"))
```

ps_regions_eval *Evaluate region numbers*

Description

This function compares multiple potential values for k, the number of clusters in to use in ps_regions(), to help you decide how well different numbers of regions fit your data set. For each value of k, it performs a cluster analysis and calculates the proportion of total variance explained (SSE, the sum of squared pairwise distances explained). It also calculates second-order metrics of the relationship between k and SSE. While many data sets have no optimal value of k and the choice is often highly subjective, these evaluation metrics can help you identify potential points where the variance explained stops increasing quickly as k increases.

Usage

```
ps_regions_eval(ps, k = 1:20, plot = TRUE, ...)
```

Arguments

ps	A phylospatial object. Must contain a dissim component generated by ps_add_dissim.
k	Vector of positive integers giving possible values for k. Values greater than the number of sites in the data set will be ignored.
plot	Logical indicating whether to print a plot of the results (TRUE, the default) or return a data frame of the results (FALSE).
...	Further arguments passed to ps_regions.

Value

The function generates a data frame with the following columns. If plot = FALSE the data frame is returned, otherwise the function prints a plot of the latter variables as a function of k:

- "k": The number of clusters.
- "sse": The proportion of total variance explained, with variance defined as squared pairwise community phylogenetic dissimilarity between sites.
- "curvature": The local second derivative. Lower (more negative) values indicate more attractive break-point values of k.
- "dist11": The distance from the point to the 1:1 line on a plot of k vs sse in which k values over the interval from 1 to the number of sites are rescaled to the unit interval. Higher values indicate more attractive values for k.

Examples

```
ps <- ps_add_dissim(ps_simulate())
ps_regions_eval(ps, k = 1:15, plot = TRUE)
```

ps_rgb *Map phylospatial data onto RGB color bands*

Description

Perform an ordination that reduces a spatial phylogenetic data set into three dimensions that can be plotted as the RGB bands of color space to visualize spatial patterns of community phylogenetic composition. This function is a wrapper around `ps_ordinate()`.

Usage

```
ps_rgb(ps, method = c("nmds", "cmds", "pca"), trans = identity, spatial = TRUE)
```

Arguments

`ps`	A `phylospatial` object with a non-null `dissim` component, generated by ps_add_dissim.
`method`	Ordination method, either "pca" (principal component analysis implemented via `stats::prcomp()`), "cmds" (classical MDS, implemented via `stats::cmdscale()`), or "nmds" (the default, nonmetric MDS, implemented via `vegan::metaMDS()`; this is slower but often preferred).
`trans`	A function giving a transformation to apply to each dimension of the ordinated data. The default is the identity function. Specifying `rank` generates a more uniform color distribution.
`spatial`	Logical indicating whether a spatial object (inherited from `ps`) should be returned. Default is TRUE.

Value

A matrix or spatial object with three variables containing RGB color values in the range 0-1.

Examples

```
ps <- ps_add_dissim(ps_simulate(50, 20, 20))
RGB <- ps_rgb(ps, method = "cmds")
terra::plotRGB(RGB * 255, smooth = FALSE)
```

ps_simulate *Simulate a toy spatial phylogenetic data set*

Description

This function generates a simple phylospatial object that can be used for testing other functions in the package. It is not intended to be realistic.

Usage

```
ps_simulate(
  n_tips = 10,
  n_x = 20,
  n_y = 20,
  data_type = c("probability", "binary", "abundance"),
  spatial_type = c("raster", "none"),
  seed = NULL
)
```

Arguments

n_tips	Number of terminals on phylogeny.
n_x	Number of raster cells in x dimension of landscape.
n_y	Number of raster cells in y dimension of landscape.
data_type	Community data type for simulated ranges: either "probability" (default), "binary", or "abundance".
spatial_type	Either "raster" or "none".
seed	Optional integer to seed random number generator.

Value

phylospatial object, comprising a random phylogeny and community matrix in which each terminal has a circular geographic range with a random radius and location. The spatial reference data is a SpatRaster.

Examples

```
# using all the defaults
ps_simulate()

# specifying some arguments
plot(ps_simulate(n_tips = 50, n_x = 30, n_y = 40, data_type = "abundance"), "comm")
```

`ps_suggest_n_iter` *Suggest number of iterations for null model convergence*

Description

Estimates the number of iterations needed for the null model randomization in `ps_rand()` to reach its stationary distribution, given a dataset and algorithm. This is a convenience wrapper around `nullcat::suggest_n_iter()` that extracts the appropriate community matrix from a `phylospatial` object. Use this before running `ps_rand()` with `fun = "nullcat"` or `fun = "quantize"` to choose an appropriate value for `n_iter`. The function runs multiple independent chains of the randomization algorithm, records a mixing diagnostic at each step, and identifies the point at which chains stabilize.

Usage

```
ps_suggest_n_iter(
  ps,
  fun = c("nullcat", "quantize"),
  plot = TRUE,
  site_groups = NULL,
  ...
)
```

Arguments

`ps`	A `phylospatial` object.
`fun`	Character: `"nullcat"` or `"quantize"`, matching the intended `fun` argument to `ps_rand()`.
`plot`	Logical: if `TRUE`, plot the mixing traces with the suggested burn-in marked.
`site_groups`	Optional grouping of sites, as passed to `ps_rand()`. Randomizations then run independently within groups, with `n_iter` applied to each group. The trace runs the grouped randomization as a whole (using the first column of a matrix of groupings), so the result estimates the `n_iter` needed per group, which can be more or fewer than the full matrix needs. Use the same `site_groups` (e.g. tilings from `ps_tile_groups()` with the same `spacing`) as in the intended `ps_rand()` call. Each traced chain costs as much as one grouped randomization of `n_iter` iterations.
`...`	Additional arguments passed to `nullcat::suggest_n_iter()` and through to `nullcat::trace_cat()`, such as `method`, `n_iter`, `n_chains`, `n_strata`, `fixed`, etc.

Value

An integer giving the suggested minimum number of iterations, with additional diagnostic information as attributes. See `nullcat::suggest_n_iter()` for details.

See Also

ps_trace(), ps_rand()

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
  set.seed(123)

  # binary data with nullcat
  ps_bin <- ps_simulate(data_type = "binary")
  ps_suggest_n_iter(ps_bin, fun = "nullcat", method = "curvecat",
                    n_iter = 5000, n_chains = 3, plot = TRUE)

  # quantitative data with quantize
  ps <- ps_simulate(data_type = "prob")
  ps_suggest_n_iter(ps, fun = "quantize", method = "curvecat",
                    n_strata = 4, fixed = "cell",
                    n_iter = 2000, n_chains = 3, plot = TRUE)
}
```

---

ps_tile_groups *Random spatial tilings of a phylospatial data set*

Description

Generates random tilings of the occupied sites in a phylospatial object, for use as site_groups in ps_rand(). This is a wrapper around nullcat::tile_groups() that extracts site coordinates from ps.

Usage

```
ps_tile_groups(ps, spacing = NULL, n = 1, group_size = NULL)
```

Arguments

ps	A phylospatial object with spatial data.
spacing	The size of the tiles: the distance between the centers of neighboring hexagonal tiles, or on the sphere, the equivalent spacing for tiles of the same area. In meters if ps has a coordinate reference system, otherwise in coordinate units. Either a single value or a vector of n values (one per tiling). Supply either spacing or group_size.
n	Number of tilings to generate. For ps_rand(), use one per randomization (n = n_rand).
group_size	Alternative to spacing: the nominal number of sites per tile, ignoring edges and gaps (see details). Either a single value or a vector of n values. Requires sites with areas (raster cells or polygons).

Details

Supplying the result to `ps_rand(site_groups = ...)` gives a spatially constrained null model: in each randomization, the community matrix is randomized independently within each tile of a different, randomly placed and oriented tiling, so taxa are only redistributed among nearby sites. Across randomizations, two sites a distance `d` apart share a tile with a probability that depends only on `d`: it falls from 1 at `d = 0` to about 0.5 at `d = 0.42 * spacing`, and to 0 at about `1.15 * spacing` in the plane or `1.3 * spacing` on the sphere. See `nullcat::tile_groups()` for details.

Site coordinates are cell centers for raster data, or feature centroids for `sf` data. If `ps` has a coordinate reference system, sites are projected to longitude and latitude and tiled on the sphere, with `spacing` in meters, so results don't depend on the map projection. Tiles on the sphere all have the same area, but vary in shape between irregular hexagons and near-squares. Without a coordinate reference system, sites are tiled with regular hexagons in the plane, with `spacing` in coordinate units.

The scale of the tiling can be set either as a distance (`spacing`) or as a nominal number of sites per tile (`group_size`). `group_size` is converted to the spacing that gives tiles of `group_size` times the average area of the occupied sites, i.e. the expected number of sites in a tile that lies wholly within a fully occupied part of the study area. Note that this ties the distance scale of the null model to the resolution of the data: the same `group_size` covers a larger distance on a coarser grid. The spacing used is returned as the `"spacing"` attribute of the result.

Sites near edges and gaps tend to fall in tiles with fewer sites than `group_size`, and a site alone in its tile is not changed by fixed-margin randomizations. Use `ps_group_sizes()` to map each site's realized average pool size.

Value

An integer matrix with a row for every occupied site (in the order of `ps$occupied`) and a column for every tiling, giving each site's tile, with the tile spacing used as its `"spacing"` attribute.

See Also

`ps_group_sizes()`, `ps_rand()`, `nullcat::tile_groups()`

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
  ps <- moss()
  tiles <- ps_tile_groups(ps, spacing = 200e3, n = 100)
  terra::plot(ps_group_sizes(ps, tiles))

  # tiles of about 40 sites each (away from edges and gaps)
  tiles40 <- ps_tile_groups(ps, group_size = 40, n = 100)
  attr(tiles40, "spacing") # in meters
}
```

ps_trace *Trace diagnostics for null model mixing*

Description

Runs multiple independent chains of the null model randomization algorithm and records a mixing diagnostic at each step, producing trace plots to assess convergence. This is a convenience wrapper around `nullcat::trace_cat()` that extracts the appropriate community matrix from a `phylospatial` object.

Usage

```
ps_trace(
  ps,
  fun = c("nullcat", "quantize"),
  plot = TRUE,
  site_groups = NULL,
  ...
)
```

Arguments

`ps`	A `phylospatial` object.
`fun`	Character: `"nullcat"` or `"quantize"`, matching the intended `fun` argument to `ps_rand()`.
`plot`	Logical: if `TRUE`, plot the traces.
`site_groups`	Optional grouping of sites, as passed to `ps_rand()`. Randomizations then run independently within groups, with `n_iter` applied to each group. The trace runs the grouped randomization as a whole (using the first column of a matrix of groupings), so the result estimates the `n_iter` needed per group, which can be more or fewer than the full matrix needs. Use the same `site_groups` (e.g. tilings from `ps_tile_groups()` with the same `spacing`) as in the intended `ps_rand()` call. Each traced chain costs as much as one grouped randomization of `n_iter` iterations.
`...`	Additional arguments passed to `nullcat::trace_cat()`, such as `method`, `n_iter`, `thin`, `n_chains`, `n_strata`, `fixed`, `stat`, etc.

Value

An object of class `"cat_trace"`. See `nullcat::trace_cat()` for details.

See Also

`ps_suggest_n_iter()`, `ps_rand()`

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
  ps_bin <- ps_simulate(data_type = "binary")
  tr <- ps_trace(ps_bin, fun = "nullcat", method = "curvecat",
                 n_iter = 2000, n_chains = 5, plot = TRUE)

  ps <- ps_simulate(data_type = "prob")
  tr <- ps_trace(ps, fun = "quantize", method = "curvecat",
                 n_strata = 4, fixed = "cell",
                 n_iter = 2000, n_chains = 5, plot = TRUE)
}
```

`quantize` *Stratified randomization of community matrix*

Description

This is a simple wrapper around `nullcat::quantize()`, included in `phylospatial` mainly for backward compatibility.

Usage

```
quantize(x = NULL, ...)
```

Arguments

`x`	Community matrix with species in rows, sites in columns, and nonnegative quantities in cells.
`...`	Additional arguments passed to `nullcat::quantize()`.

Details

The nullcat quantize routine involves three steps: converting a quantitative matrix to categorical strata, permuting the resulting categorical matrix using one of several categorical null model algorithms, and mapping the randomized categories back to quantitative values. Supply arguments via `...` to control options for each of these stages.

Value

A randomized version of `x`.

Examples

```
if (requireNamespace("nullcat", quietly = TRUE)) {
      # example quantitative community matrix
      comm <- matrix(runif(2500), 50)

      # examples of different quantize usage
      rand <- quantize(comm)
      rand <- quantize(comm, n_strata = 4, transform = sqrt, fixed = "row")
      rand <- quantize(comm, method = "swapcat", n_iter = 500)
}
```

---

to_spatial *Convert a site-by-variable matrix into a SpatRaster or sf object*

Description

Convert a site-by-variable matrix into a SpatRaster or sf object

Usage

```
to_spatial(m, template)
```

Arguments

m	Matrix or vector with the same number of rows as sites in `template`. Note that `ps$comm` contains only occupied sites and must be expanded with `ps_expand()` before passing to this function.
template	`SpatRaster` layer with number of cells equal to the number of rows in m, or `sf` data frame with same number of rows as m.

Value

`SpatRaster` with a layer for every column in `m`, or `sf` data frame with a variable for every column in `m`, depending on the data type of `template`.

Examples

```
ps <- moss()
# ps$comm contains only occupied sites, so expand before converting:
to_spatial(ps_expand(ps, ps$comm[, 1:5]), ps$spatial)
```

`tree_scaling` *Scale branch lengths on a phylogenetic tree*

Description

A family of functions to modify the branch lengths of a phylogeny in ways relevant to spatial phylogenetic analyses. These fall into two conceptual categories. *Affine rescaling* (`rescale_tree`) applies a single multiplicative factor to every branch, changing units without changing relative branch lengths. *Differential transforms* (`slice_tree`, `delta_tree`, `uniform_tree`) change the shape of the branch-length distribution by treating different branches differently based on their position or length.

Usage

```
rescale_tree(tree, method = c("raw", "sum1", "tip1"))

slice_tree(tree, min = -Inf, max = Inf, reference = NULL, from = "tips")

delta_tree(tree, delta)

uniform_tree(tree)
```

Arguments

`tree`	Phylogeny of class ape::phylo.
`method`	Character giving the rescaling method for `rescale_tree`: `"raw"`, `"sum1"`, or `"tip1"`.
`min`, `max`	Numeric giving the lower and upper bounds of the depth window for slicing, in the same units as `reference` if provided, otherwise in the units of `tree$edge.length`. Defaults are `-Inf` and `Inf` (no slicing). See `from` for the meaning of "depth."
`reference`	Optional phylogeny of class ape::phylo whose branch lengths define the slicing window. Must share topology with `tree`. If `NULL` (default), the window is defined on `tree`'s own branch lengths. When supplied, slicing identifies the fraction of each reference-branch's extent that falls within the window, and applies that fraction to `tree`'s branch length. This allows slicing a phylogram by a chronogram, for example.
`from`	Character: `"tips"` (default) or `"root"`. Determines whether `min` and `max` are measured from the tips (so `min = 10, max = 30` selects branch portions 10 to 30 units before the deepest tip) or from the root (so `min = 10, max = 30` selects branch portions 10 to 30 units after the root).
`delta`	Numeric exponent for `delta_tree`. Must be positive. For the limiting case `delta = 0`, use `uniform_tree` instead.

Details

The two categories of operations serve different purposes and can be composed. A typical workflow is to apply a differential transform first (if any), and then optionally an affine rescaling (if any).

`slice_tree` retains only the portions of each branch that fall within a specified window along the tree, setting other portions to zero. This implements the "time-sliced" approach to phylodiversity analysis, enabling questions about divergence within a specific depth range. The window can be defined on the tree itself, or on a separate `reference` tree with identical topology (e.g., slicing a phylogram by a chronogram, to isolate molecular divergence within a specific age range). See Araujo et al. (2024) for discussion of related approaches. Note that slicing modifies branch lengths, which changes the depth coordinate system of the resulting tree (because zeroed-out branches near the root collapse, shifting the retained portion toward the root). If you need to apply multiple operations that depend on the original tree's depth structure, pass the original tree as `reference` on subsequent calls.

`delta_tree` applies Pagel's (1999) delta transformation, raising each node depth (root-to-node distance) to the power `delta`, then recomputing branch lengths from the transformed depths. The resulting tree is then rescaled to match the original tree's total height, so that only the *distribution* of branch length along the tree changes, not the total. Values of `delta < 1` lengthen deep branches at the expense of shallow ones, emphasizing deeper divergence; values of `delta > 1` do the opposite, emphasizing recent divergence. Delta is the continuous analog of `slice_tree`: both operate on position within the tree rather than on individual branch lengths. The transformation is most interpretable on ultrametric trees, where node depths have a single consistent meaning. See Saladin et al. (2019) for an application to spatial phylogenetics.

`uniform_tree` sets all branch lengths equal to 1, producing a cladogram. Phylogenetic diversity computed on a uniform tree is equivalent to clade richness (CR), giving equal weight to every splitting event.

`rescale_tree` applies an affine rescaling:

- `"raw"`—no rescaling (identity operation).
- `"sum1"`—divides all branch lengths so they sum to 1. Branch lengths then represent fractions of total tree length, and PD values represent the fraction of total evolutionary history present in a site.
- `"tip1"`—divides all branch lengths by the longest root-to-tip path, so that the longest path equals 1. For ultrametric trees, every tip has path length 1 (following Pavoine et al. 2005). For non-ultrametric trees, only the longest path equals 1.

Rescaling does not change relative branch lengths, so it does not affect spatial patterns, correlations, or statistical significance of diversity metrics—only their numeric scale. Differential transforms, by contrast, change what the metrics are measuring.

Value

A phylogeny of class ape::phylo with modified branch lengths. Topology is always preserved; `tree$tip.label`, `tree$node.label`, `tree$edge`, and `tree$Nnode` are unchanged.

References

Araujo, M. L., Silva, V., and Cassemiro, F. A. S. (2024). 'treesliceR': a package for slicing phylogenies and inferring phylogenetic patterns over evolutionary time. Ecography, e07364.

Pagel, M. (1999). Inferring the historical patterns of biological evolution. Nature, 401(6756), 877-884.

Pavoine, S., Ollier, S., and Dufour, A.-B. (2005). Is the originality of a species measurable? Ecology Letters, 8(6), 579-586.

Saladin, B., Thuiller, W., Graham, C. H., Lavergne, S., Maiorano, L., Salamin, N., and Zimmermann, N. E. (2019). Environment and evolutionary history shape phylogenetic turnover in European tetrapods. Nature Communications, 10(1), 249.

Examples

```
moss_tree <- ape::read.tree(system.file(
      "extdata", "moss_tree.nex", package = "phylospatial"))
par(mfrow = c(1, 4), mar = c(2, 1, 2, 1))

plot(moss_tree,
     show.tip.label = FALSE, main = "original")
plot(uniform_tree(moss_tree),
     show.tip.label = FALSE, main = "cladogram")
plot(slice_tree(rescale_tree(moss_tree, "tip1"),
                min = 0, max = 0.25, from = "tips"),
     show.tip.label = FALSE, main = "sliced (recent 25%)")
plot(delta_tree(moss_tree, delta = 1/3),
     show.tip.label = FALSE, main = "delta-transformed")
```

Index

ape::phylo, *52, 53*

benefit, 2
benefit(), *30*, *34*

cellSize, *11*
clade_dist, 3
commsim, *39*

delta_tree *(*tree_scaling*)*, 52
delta_tree(), *19*
designdist, *15*

expanse, *11*

graphics::plot(), *29*

hclust, *42*

make.commsim, *13*
moss, 4

nullcat, *39, 40*
nullcat::nullcat(), *40*
nullcat::suggest_n_iter(), *46*
nullcat::tile_groups(), *47, 48*
nullcat::trace_cat(), *46*, *49*
nullcat_methods, *39*
nullmodel, *39*

perim, *11*
phylo, *5*
phylospatial, 5
phylospatial(), *10, 11*, *15*, *19*, *21*, *25, 26*, *33*, *37*, *40*
plot, *9*
plot.phylo, *9*
plot.phylospatial, 8
plot.ps_performance *(*ps_performance*)*, 29
plot.sf, *9*
plot_lambda, 9, *32*
plot_lambda(), *34*
prioritizr::add_boundary_penalties(), *36*
prioritizr::add_max_n_targets_met_objective(), *35*
prioritizr::add_max_phylo_div_objective(), *36*
prioritizr::add_max_phylo_end_objective(), *36*
prioritizr::add_min_set_objective(), *35*
prioritizr::add_min_shortfall_objective(), *35*
prioritizr::eval_feature_representation_summary(), *36*
prioritizr::problem(), *36*
prioritizr::solve(), *34*
ps_add_dissim, 10, *28*, *42–44*
ps_add_dissim(), *16*
ps_area, 10
ps_area(), *6*
ps_canape, 11
ps_canape(), *8*, *13*, *41*
ps_canaper, 13
ps_canaper(), *8*
ps_dissim, *10*, 14
ps_dissim(), *8*, *17–19*, *23*
ps_dissim_focal, 16
ps_dissim_focal(), *16*, *19*
ps_dissim_paired, 18
ps_dissim_paired(), *16, 17*
ps_diversity, 20, *38, 39*
ps_diversity(), *8*, *41*
ps_expand, 22
ps_expand(), *22*, *35*
ps_geodist, 23
ps_get_comm, 24
ps_grid, 25
ps_grid(), *7, 8*

ps_group_sizes, 27
ps_group_sizes(), *48*
ps_ordinate, 28
ps_performance, 29
ps_performance(), *8*, *34, 35*
ps_prioritize, *9*, 31
ps_prioritize(), *4*, *8*, *29, 30*, *35, 36*
ps_prioritizr, 34
ps_prioritizr(), *8*, *34*
ps_quantize, 37
ps_rand, 38
ps_rand(), *8*, *12, 13*, *28*, *37, 38*, *46–49*
ps_regions, *14*, 41, *43*
ps_regions(), *8*
ps_regions_eval, *42*, 43
ps_rgb, *14*, 44
ps_rgb(), *29*
ps_simulate, 45
ps_suggest_n_iter, 46
ps_suggest_n_iter(), *39, 40*, *49*
ps_tile_groups, 47
ps_tile_groups(), *27, 28*, *40, 41*, *46*, *49*
ps_trace, 49
ps_trace(), *47*

quantize, *37*, *39, 40*, *50*, 50
quantize_prep, *39*

rescale_tree *(*tree_scaling*)*, 52
rescale_tree(), *6*

simulate.nullmodel, *39, 40*
slice_tree *(*tree_scaling*)*, 52
SpatRaster, *5*, *10*

terra::crs(), *26*
terra::plot(), *12*
terra::writeRaster(), *12*
to_spatial, 51
tree_scaling, 52

uniform_tree *(*tree_scaling*)*, 52

vegan::designdist2(), *17*
vegdist, *15*