Package: dissmapr (via r-universe)

May 31, 2026

Type Package

Title Workflow for Compositional Dissimilarity and Biodiversity Turnover Analysis

Version 0.1.0

Description Tools to compute and map biodiversity turnover at the macroscale.

License MIT + file LICENSE

URL https://github.com/b-cubed-eu/dissmapr, https://b-cubed-eu.github.io/dissmapr/

BugReports https://github.com/b-cubed-eu/dissmapr/issues

Suggests devtools, knitr, pkgdown, remotes, rmarkdown, testthat (>= 3.0.0)

Imports caret, cluster, clValid, corrplot, data.table, dplyr, entropy, factoextra, fields, future, geodata, geosphere, ggplot2, httr, mclust, NbClust, patchwork, pbapply, purrr, rcmdcheck, rlang, reshape2, sf, terra, tidyr, tidyterra, vegan, viridis, zoo, zetadiv

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 3.5)

Repository https://b-cubed-eu.r-universe.dev

Date/Publication 2026-04-01 15:03:12 UTC

RemoteUrl https://github.com/b-cubed-eu/dissmapr

RemoteRef HEAD

RemoteSha 3b9b9d798d65d75ba667ee2ce44c3d5397ca3fdb

Contents

---

abund *Calculate Species Abundance*

Description

Sums total individuals per site; reports absolute pairwise differences and dispersion metrics for higher orders.

Usage

```
abund(vec_from, vec_to = NULL)
```

Arguments

`vec_from`	A numeric vector representing species counts at the first site.
`vec_to`	(Optional) A numeric vector for pairwise or higher-order comparisons.

Value

The species abundance value.

`assign_mapsheet` *Add Nearest Mapsheet Code and Center Coordinates*

Description

This function takes an existing data frame with coordinate columns and appends:

1. the nearest mapsheet code (format "E|WDDDN|SDDBB"), and
2. the center longitude/latitude of that mapsheet.

Usage

```
assign_mapsheet(
  data,
  x_col = "x",
  y_col = "y",
  cell_size = 1,
  unit = c("deg", "min", "sec", "m")
)
```

Arguments

`data`	A data.frame containing at least the coordinate columns.
`x_col`	Character. Name of the longitude (x) column. Default `"x"`.
`y_col`	Character. Name of the latitude (y) column. Default `"y"`.
`cell_size`	Numeric. Cell size: e.g. 1 for 1deg cells, 15 for 15' cells.
`unit`	Character. Unit of `cell_size`; one of: • `"deg"` for decimal degrees (default) • `"min"` for arc-minutes • `"sec"` for arc-seconds • `"m"` for metres (projected coords).

Details

It divides the world into regular grid cells (in degrees, minutes, seconds, or metres), finds which cell each point falls into, and then formats the sheet code based on the cell centre.

Value

The input `data` with three new columns:

- `mapsheet`: the 7-character sheet code
- `center_lon`: centre longitude of that sheet
- `center_lat`: centre latitude of that sheet

Examples

```
bird_obs = data.frame(
  site_id = 1:6,
  x = c(22.71862, 20.40034, 18.51368, 18.38477, 23.56160, 18.87285),
  y = c(-33.98912, -34.45408, -34.08271, -34.25946, -33.97620, -34.06225),
  species = c("Fulica cristata", "Numida meleagris coronatus",
              "Anas undulata", "Oenanthe familiaris",
              "Cyanomitra veroxii", "Gallinula chloropus"),
  value = rep(1, 6),
  year = c(2023, 2022, 2023, 2016, 2016, 2019),
  month = c(5, 12, 10, 8, 9, 2),
  day = c(12, 4, 3, 1, 4, 21)
)

# For 1deg mapsheet cells:
bird_obs2 = assign_mapsheet(bird_obs, cell_size = 1, unit = "deg")
head(bird_obs2)
```

`compute_orderwise` *Compute Order-wise Metrics*

Description

This function computes metrics for ecological data across specified order levels. It supports single-site, pairwise, and higher-order calculations, and allows for parallel processing for efficiency.

Usage

```
compute_orderwise(
  df,
  func,
  site_col,
  sp_cols = NULL,
  order = 2,
  sample_no = NULL,
  sample_portion = 1,
  parallel = TRUE,
  n_workers = parallel::detectCores() - 1
)
```

Arguments

`df`	A data frame containing the ecological data.
`func`	A function to compute metrics. It must accept inputs in the form of species vectors or site information depending on the order.
`site_col`	A character string specifying the column name in `df` representing site IDs.

sp_cols	A vector of column names in df representing species data (default: NULL).
order	An integer or vector of integers specifying the order(s) of computation. • 1: Single-site computations. • 2: Pairwise computations. • >= 3: Higher-order computations.
sample_no	An integer specifying the maximum number of combinations to sample for higher-order computations (default: NULL for all combinations).
sample_portion	A numeric value between 0 and 1 indicating the proportion of combinations to sample for higher-order computations (default: 1, meaning 100%).
parallel	A logical value indicating whether to enable parallel computation (default: TRUE).
n_workers	An integer specifying the number of parallel workers to use (default: one less than the number of available cores).

Value

A data.table containing the results of computations. Columns include:

- site_from: The source site.
- site_to: The target site(s) (NA for order = 1).
- order: The computation order.
- value: The computed metric value.

Examples

```
# Example usage with a custom metric function
metric_func = function(vec1, vec2 = NULL) {
  if (is.null(vec2)) {
    return(sum(vec1))  # Example: species richness
  } else {
    return(sum((vec1 - vec2)^2))  # Example: pairwise dissimilarity
  }
}

data = data.frame(
  site = rep(letters[1:3], each = 3),
  sp1 = c(1, 0, 2, 1, 2, 0, 0, 1, 1),
  sp2 = c(0, 1, 1, 2, 0, 0, 1, 0, 2)
)

# SPECIES RICHNESS
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
richness = function(vec_from, vec_to = NULL) {
 if (is.null(vec_to)) {
   # Handle single-site calculations (order = 1)
   return(sum(vec_from != 0, na.rm = TRUE))
 } else if (length(vec_from) > 1 && length(vec_to) > 1) {
   # Handle pairwise or higher-order comparisons
   return(abs(sum(vec_from != 0, na.rm = TRUE) - sum(vec_to != 0, na.rm = TRUE)))
```

```
 } else {
   # Invalid input case
   return(NA)
 }
}

rich_o12 = compute_orderwise(
 df = block_sp,
 func = richness,
 site_col = 'grid_id',
 sp_cols = sp_cols,
 # sample_no = 1000,
 # sample_portion = 0.5,  # Default is 1 (100%)
 order = 1:2,  # Compute for pairwise and higher-order comparisons
 parallel = TRUE,
 n_workers = 4
)
head(rich_o12)
```

`cor_pears` *Calculate Pearson's Correlation*

Description

Pearson's correlation for linear abundance relationships; aggregated across sites.

Usage

```
cor_pears(vec_from, vec_to)
```

Arguments

`vec_from`	A numeric vector representing species abundances at the first site.
`vec_to`	A numeric vector representing species abundances at the second site.

Value

Pearson's correlation coefficient.

cor_spear *Calculate Spearman's Rank Correlation*

Description

Spearman's rank correlation for abundances; pairwise and averaged multi-site.

Usage

```
cor_spear(vec_from, vec_to)
```

Arguments

vec_from	A numeric vector representing species abundances at the first site.
vec_to	A numeric vector representing species abundances at the second site.

Value

Spearman's rank correlation coefficient.

diss_bcurt *Calculate Bray-Curtis Dissimilarity*

Description

Bray-Curtis dissimilarity on abundances; 0-1 for pairwise, averages or totals for multi-site.

Usage

```
diss_bcurt(vec_from, vec_to)
```

Arguments

vec_from	A numeric vector representing species abundances at the first site.
vec_to	A numeric vector for species abundances at the second site.

Value

The Bray-Curtis dissimilarity value.

dissmapr *dissmapr: Workflow for Compositional Dissimilarity & Biodiversity Turnover Analysis*

Description

dissmapr is an R package for analysing compositional dissimilarity and biodiversity turnover across spatial gradients.

format_df *Format Biodiversity Records to Long / Wide*

Description

Converts a table of biodiversity observations between **long** and **wide** layouts while standardising key column names.

- **Long format** - one row per observation with columns site_id, x, y, species, value (+ optional extra_cols).
- **Wide format** - one row per site with species as individual columns.

Usage

```
format_df(
  data,
  format = NULL,
  x_col = NULL,
  y_col = NULL,
  site_id_col = NULL,
  species_col = NULL,
  value_col = NULL,
  sp_col_range = NULL,
  extra_cols = NULL
)
```

Arguments

data	A data frame containing biodiversity records.
format	Character; target layout "long" or "wide". If NULL the format is inferred automatically.
x_col, y_col	Character. Names of the longitude (x) and latitude (y) columns. If NULL, common alternatives are searched.
site_id_col	Character. Column giving a unique site identifier. If NULL, a new site_id is generated from the coordinate pair.

species_col	Character. Column containing species names (required for format = "long").
value_col	Character. Column with numeric values such as presence/absence (0/1) or abundance. If NULL, each record is assigned a value of 1.
sp_col_range	Integer vector giving the index of species columns when format = "wide". If NULL all non-coordinate / non-metadata columns are treated as species.
extra_cols	Character vector of additional columns to carry through to the output (e.g. sampling metadata or environmental covariates).

Details

If column names are not supplied, the function attempts to detect common variants (e.g. "lon", "longitude" for *x*). When converting long -> wide, duplicate observations of the same species at a site are aggregated by summing value. When converting wide -> long, species columns are inferred either from sp_col_range or by excluding coordinate / metadata columns.

Value

A named list with up to two elements

- site_obs - a long-format data frame (returned only when format = "long").
- site_spp - a wide site x species data frame.

Dependencies

Relies on **dplyr**, **tidyr**, and **rlang** (loaded with requireNamespace()).

See Also

group_by, pivot_wider

Examples

```
## --- Example 1: long  ->  wide -------------------------------------------
ex_long <- data.frame(
  lon     = c(23.10, 23.10, 23.25, 23.25),
  lat     = c(-34.00, -34.00, -34.05, -34.05),
  species = c("sp1",  "sp2",  "sp1",  "sp3"),
  count   = c(1, 2, 3, 1)
)

out_long <- format_df(
  data        = ex_long,
  format      = "long",
  x_col       = "lon",
  y_col       = "lat",
  species_col = "species",
  value_col   = "count"
)

head(out_long$site_spp)
```

```
## --- Example 2: wide ->  long -------------------------------------------
ex_wide <- out_long$site_spp

out_wide <- format_df(
  data   = ex_wide,
  format = "wide"
)

head(out_wide$site_spp)
```

`generate_grid` *Generate Spatial Grid and Gridded Summaries*

Description

Builds a regular lattice over a point-occurrence data set, assigns every record a `grid_id`, and returns grid-level summaries in both abundance and presence/absence formats.

Usage

```
generate_grid(
  data,
  x_col = "x",
  y_col = "y",
  grid_size = 0.5,
  sum_cols = NULL,
  extra_cols = NULL,
  crs_epsg = 4326,
  unit = c("deg", "min", "sec", "m")
)
```

Arguments

`data`	Data frame of points with x-y coordinates.
`x_col`	y_col Column names for longitude and latitude.
`grid_size`	Cell size (degrees or metres, depending on CRS).
`sum_cols`	Character or numeric vector of columns to aggregate. Note: Numeric indices are converted to names internally.
`extra_cols`	Additional columns to keep (optional).
`crs_epsg`	EPSG code of the coordinate reference system.
`unit`	One of "deg", "min", "sec", or "m".

Details

The function:

1. Tiles the study extent with square cells of size `grid_size`.
2. Computes cell centroids and, for geographic CRS data, optional mapsheet codes (1:250 000 "BB" series).
3. Aggregates user-specified columns (`sum_cols`) per cell, producing
   - `grid_spp` - counts / abundances
   - `grid_spp_pa` - binary 0 / 1 table for dissimilarity analyses.
4. Calculates helper metrics (`obs_sum`, `spp_rich`) for each cell.
5. Rasterises key layers (`grid_id`, `obs_sum`, `spp_rich`) and preserves any extra metadata supplied via `extra_cols`.

Value

A list containing

- `grid_r` - multi-layer **SpatRaster**
- `grid_sf` - polygon lattice with centroids & metrics
- `grid_spp` - abundance summary (*data.frame*)
- `grid_spp_pa` - presence/absence summary (*data.frame*)

Examples

```
set.seed(123)
data = data.frame(
  x = runif(100, -10, 10),
  y = runif(100, -10, 10),
  species1 = rpois(100, 5),
  species2 = rpois(100, 3),
  recordedBy = sample(LETTERS, 100, replace = TRUE)
)
grid_result = generate_grid(data, x_col = "x", y_col = "y",
                            grid_size = 1, sum_cols = 3:4,
                            extra_cols = c("recordedBy"))
print(grid_result$block_sp)
plot(grid_result$grid_sf["grid_id"])
```

`geodist_helper` *Calculate geographic distance via Haversine formula (helper)*

Description

Computes the geographic (Haversine) distance in meters between two coordinate pairs, or returns 0 for single-site (order=1) calculations when `vec_to` is `NULL`.

Usage

```
geodist_helper(
  vec_from,
  vec_to = NULL,
  coord_cols = c("centroid_lon", "centroid_lat")
)
```

Arguments

vec_from	Numeric vector of length 2, or a named vector containing coordinates. If named, should include names given by coord_cols.
vec_to	Optional numeric vector of length 2 (destination coordinates); if NULL (default), the function returns 0.
coord_cols	Character vector of length 2 giving the names of the longitude and latitude elements in vec_from/vec_to when those are named vectors. Defaults to c("x", "y").

Value

Numeric distance in meters between the two points, or 0 if vec_to is NULL.

---

get_enviro_data *Retrieve, crop, resample, and link environmental rasters to sampling sites*

Description

get_enviro_data() automates six steps:

1. **AOI build** - buffers the convex hull of all input points.
2. **Raster acquisition** - downloads or loads a multi-layer stack (WorldClim, SoilGrids, footprint, population, or user-supplied).
3. **Cropping** - trims the stack to the buffered AOI.
4. **Optional resampling** - resamples the cropped stack to a user-defined grid resolution.
5. **Extraction & gap fill** - pulls raster values at each site and linearly interpolates isolated NAs.
6. **Assembly** - returns a tidy *site x environment* table plus the cropped (and resampled) raster and an sf layer of sites.

Usage

```
get_enviro_data(
  data,
  buffer_km = 10,
  source = "geodata",
  var = "bio",
```

```
  res = 2.5,
  grid_r = NULL,
  path = "data/",
  year = NULL,
  depth = NULL,
  stat = "mean",
  model = NULL,
  ssp = NULL,
  time = NULL,
  sp_cols = NULL,
  ext_cols = NULL
)
```

Arguments

data	Data frame of spatial points; must include lon/lat columns such as "x","y" or "centroid_lon","centroid_lat".
buffer_km	Buffer width (km) for the AOI. Default 10.
source	"geodata" (default) to fetch layers via **geodata** or "local" to supply a local SpatRaster or file path.
var	Raster product to download (see details) or ignored when source = "local".
res	Resolution (arc-min) for WorldClim/WorldPop layers (geodata).
grid_r	Optional grid resolution (in same CRS units) to which the cropped raster is resampled before extraction. If NULL, no resampling is performed.
path	Cache folder for downloaded rasters (created if absent).
year	model, ssp, time Optional arguments for time-stamped products (human footprint, population, CMIP6, etc.).
depth	stat Arguments passed to geodata::soil_world().
sp_cols	**Columns to drop** from the final table (e.g. a large species matrix). Accepts names or numeric indices *relative to* data.
ext_cols	**Columns to append** verbatim (e.g. "obs_sum","spp_rich").

Value

A list with

- env_rast SpatRaster - cropped (and optionally resampled) environmental stack
- sites_sf sf POINT layer (WGS-84) of the input sites
- env_df Tibble with site ID, coordinates, every raster variable, plus any columns requested in ext_cols

`get_occurrence_data` *Import and harmonise biodiversity-occurrence data*

Description

`get_occurrence_data()` reads occurrence records from a **local CSV/file path**, an **in-memory** `data.frame`, or a **GBIF download (ZIP)** and returns a tidy data frame in either **long** (one row = one record) or **wide** (one row = one site, one column = one species) form.

Usage

```
get_occurrence_data(
  data = NULL,
  source_type = c("local_csv", "data_frame", "gbif"),
  gbif_zip_url = NULL,
  download_dir = tempdir(),
  sep = ",",
  site_id_col = NULL,
  x_col = NULL,
  y_col = NULL,
  sp_name_col = NULL,
  pa_col = NULL,
  abund_col = NULL,
  species_cols = NULL
)
```

Arguments

`data`	File path (when `source_type = "local_csv"`), an in-memory `data.frame` (`"data_frame"`), or `NULL` (`"gbif"`).
`source_type`	`"local_csv"`, `"data_frame"`, or `"gbif"`.
`gbif_zip_url`	URL to a GBIF download ZIP (required when `source_type = "gbif"`).
`download_dir`	Folder to save the ZIP/extracted file (default: `tempdir()`).
`sep`	Field separator for CSVs (default `","`).
`site_id_col, x_col, y_col, sp_name_col, pa_col, abund_col`	Optional custom column names.
`species_cols`	**Optional** numeric or character vector specifying the species columns in a wide input (e.g. `4:11` or `c("Sp1","Sp2")`). Overrides the default `"sp_*"` detection.

Details

Column names are auto-detected from common patterns (`"site_id"`, `"x"`, `"y"`, `"sp_name"`, `"pa"` or `"abund"`). Supply `*_col` arguments **only** when your data use different names.

For wide data the helper normally looks for columns that start with `"sp_"`. Set `species_cols` to a numeric range (e.g. `4:11`) or a character vector of column names when the species columns do **not** follow the `"sp_*"` convention.

Value

A `data.frame`:

Long format Columns `site_id`, `x`, `y`, `sp_name`, plus `pa` *or* `abund`.

Wide - long Same columns after stacking the specified or auto-detected species columns.

Workflow

1. **Read** the data from `source_type`.
2. **Detect / insert** compulsory columns (site, coords, species, value).
3. **Validate** coordinates (-180 <= lon <= 180, -90 <= lat <= 90).
4. **Return**
   - a long table (`site_id`, `x`, `y`, `sp_name`, `pa|abund`) when species name + value columns are present; or
   - a long table reshaped from wide species columns.

See Also

`tidyr::pivot_longer()` used internally.

Examples

```
# 1. Local CSV example ------------------------------------------------
tmp <- tempfile(fileext = ".csv")
df_local <- data.frame(
  site_id = 1:10,
  x = runif(10), y = runif(10),
 sp_name = c("plant1", "plant2","plant3", "plant4","plant5", "plant1", "plant2","plant3", "plant4","plant5"),
  abun = sample(0:20, size = 10, replace = TRUE)
)
write.csv(df_local, tmp, row.names = FALSE)
local_test = get_occurrence_data(data = tmp, source_type = "local_csv", sep = ",")

# 2. Existing wide-format data.frame ----------------------------------------------
df_wide <- df_local %>%
  pivot_wider(
    names_from  = sp_name,   # these become column names
    values_from = abun,  # fill these cell values
    values_fn   = sum,       # sum duplicates
    values_fill = 0          # fill missing with 0
  )
wide_test = get_occurrence_data(data = df_wide, source_type = "data_frame", species_cols = 4:11)

# 3. Custom names -----------------------------------------------------------
names(sim_dat)[1:5] <- c("plot_id", "lon", "lat", "taxon", "presence")
occ_long2 <- get_occurrence_data(
  data           = sim_dat,
  source_type    = "data_frame",
  site_id_col    = "plot_id",
  x_col          = "lon",
```

```
  y_col          = "lat",
  sp_name_col    = "taxon",
  pa_col         = "presence"
)
head(occ_long2)

# 4. GBIF download (requires internet) ------------------------------------------------
## Not run:
gbif_test = get_occurrence_data(
  source_type   = "gbif",
 gbif_zip_url = "https://api.gbif.org/v1/occurrence/download/request/0038969-240906103802322.zip"
)

## End(Not run)
```

`map_bioreg` *Raster-based Clustering and Interpolation of Bioregional Data*

Description

`map_bioreg` performs clustering (k-means, PAM, hierarchical, GMM) on spatial point data, computes modal or user-specified interpolation (none, nearest-neighbour, TPS), aligns cluster labels by overlap, and returns both data tables and raster stacks.

Usage

```
map_bioreg(
  data,
  scale_cols,
  method = c("kmeans", "pam", "hclust", "gmm", "all"),
  k_override = NULL,
  x_col = "x",
  y_col = "y",
  interpolate = c("none", "nn", "tps", "all"),
  res = 0.5,
  crs = "EPSG:4326",
  plot = TRUE,
  bndy_fc = NULL
)
```

Arguments

`data`	A single data.frame of point observations or a named list of such data.frames (e.g. different scenarios). Each data.frame must contain coordinate columns and variables to scale.
`scale_cols`	Character vector of column names in `data` to standardize before clustering.
`method`	Character vector of clustering methods to apply; choices of "kmeans", "pam", "hclust", "gmm", or "all".

k_override	Integer. If provided, fixes the number of clusters rather than computing via silhouette.
x_col, y_col	Strings giving the names of the longitude (x) and latitude (y) columns for spatial mapping.
interpolate	One of "none", "nn", "tps", or "all"; selects which interpolation(s) to compute.
res	Numeric resolution of output rasters (in the same units as coordinates).
crs	Coordinate reference system string for raster outputs (e.g. "EPSG:4326").
plot	Logical; if TRUE, displays spatial tile plots of clusters/modes.
bndy_fc	Optional sf or SpatVector polygon to overlay.

Details

Internally, the function standardizes scale_cols, runs requested clustering(s), computes a modal consensus label, and then aligns each algorithm's cluster numbers to the k-means reference by maximal cell overlap. Three interpolation functions (fill_none_full, fill_nn_full, fill_tps_full) generate rasters of raw and aligned labels. Progress bars display per scenario.

Value

A named list with elements:

none, nn, tps Named lists of Terra SpatRaster stacks for each scenario, each layer labelled "*algn*".

table A data.frame: original points, cluster columns, modal label, aligned labels.

plots List of ggplot objects (invisible) when plot = TRUE.

methods Character vector of methods actually computed.

Examples

```
library(terra)
# simulate a single data.frame
set.seed(42)
df = data.frame(
    centroid_lon = runif(100, 16, 33),
  centroid_lat = runif(100, -35, -22),
  pred_zetaExp = rnorm(100)
)
# single scenario clustering
out1 = map_bioreg(
    data = df,
  scale_cols = c("pred_zetaExp", "centroid_lon", "centroid_lat"),
  method = "all",
  k_override = 4,
  x_col = "centroid_lon",
  y_col = "centroid_lat",
  plot = FALSE
)

# simulate multiple scenarios
```

```
df2 = df
df2$centroid_lon = df2$centroid_lon + 5
scen_list = list(current = df, future = df2)
out2 = map_bioreg(
    data = scen_list,
  scale_cols = c("pred_zetaExp", "centroid_lon", "centroid_lat"),
  method = "kmeans",
  k_override = 3,
  x_col = "centroid_lon",
  y_col = "centroid_lat",
  plot = FALSE
)
```

`map_bioregDiff` *Map Bioregional Change Metrics Between Categorical Raster Layers*

Description

Calculates five complementary indices that quantify how the categorical (e.g. bioregion / cluster) label of each raster cell changes across a temporal or scenario stack of rasters:

- **Difference count** - total number of times a cell's label differs from the first layer.
- **Shannon entropy** - information-theoretic diversity of labels within the cell's time-series.
- **Stability** - proportion of layers in which the label is identical to the first layer *(1 = always unchanged, 0 = always different)*.
- **Transition frequency** - sum of binary change maps between successive layers (*how often a change occurs between any pair of neighbours*).
- **Weighted change index** - cumulative dissimilarity-weighted change where the weight is derived from the empirical frequency of transitions between all pairs of labels.

Usage

```
map_bioregDiff(raster_input, approach = "all")
```

Arguments

`raster_input`	A multi-layer `SpatRaster` **or** a `list` of single-layer `SpatRaster` objects representing the same spatial extent/resolution.
`approach`	Character string specifying the metric to return: `"difference_count"`, `"shannon_entropy"`, `"stability"`, `"transition_frequency"`, `"weighted_change_index"`, or `"all"` (default) for a five-layer stack containing every metric.

Details

The dissimilarity weights for the **weighted change index** are built from the observed transition table of successive layers, normalised to lie between 0 and 1 (larger = rarer transition). The function accepts either a multi-layer `SpatRaster` or a plain `list` of single-layer `SpatRaster`s, which is internally concatenated with **terra**.

Value

A SpatRaster:

- **single-layer** if approach is one of the named metrics;
- **five-layer** (names: Difference_Count, Shannon_Entropy, Stability, Transition_Frequency, Weighted_Change_Index) if approach = "all".

See Also

app, rast

Examples

```
## ------------------------------------------------------------
## Minimal reproducible example with three random categorical
## rasters (four classes, identical geometry)
## ------------------------------------------------------------
if (requireNamespace("terra", quietly = TRUE)) {
  set.seed(42)

  r1 <- terra::rast(nrows = 40, ncols = 40,
                    vals  = sample(1:4, 40 * 40, TRUE))
  r2 <- terra::rast(r1, vals = sample(1:4, ncell(r1), TRUE))
  r3 <- terra::rast(r1, vals = sample(1:4, ncell(r1), TRUE))

  r_stack <- terra::rast(list(r1, r2, r3))
  names(r_stack) <- paste0("t", 1:3)

  ## 1. All five metrics
  diff_all <- map_bioregDiff(r_stack, approach = "all")
  print(diff_all)

  ## 2. Just the Shannon-entropy layer
  ent <- map_bioregDiff(r_stack, approach = "shannon_entropy")
  terra::plot(ent, main = "Shannon entropy of label sequence")
}
```

---

mutual_info *Calculate Mutual Information*

Description

Mutual information for dependence (linear + non-linear); pairwise or cumulative multi-site.

Usage

```
mutual_info(vec_from, vec_to)
```

Arguments

vec_from	A numeric or categorical vector for the first variable.
vec_to	A numeric or categorical vector for the second variable.

Value

The mutual information value between the two variables.

---

orderwise_diss_gower *Compute Gower dissimilarity between two site vectors*

Description

Calculates the Gower dissimilarity (a bounded 0-1 measure) between two numeric vectors (e.g., species abundances) using the `cluster::daisy()` function. When the second vector is `NULL` (order 1), returns `NA_real_` since a pairwise comparison is not meaningful.

Usage

```
orderwise_diss_gower(vec_from, vec_to = NULL)
```

Arguments

vec_from	Numeric vector representing the first site's attributes (e.g., species counts or trait values).
vec_to	Optional numeric vector for the second site; if `NULL`, the function returns `NA_real_`.

Value

A single numeric dissimilarity value between 0 and 1, or `NA_real_` if `vec_to` is `NULL`.

---

phi_coef *Calculate Phi Coefficient*

Description

Phi coefficient for presence-absence; -1 to +1 pairwise, summary of means/variances across combinations.

Usage

```
phi_coef(vec_from, vec_to)
```

Arguments

vec_from	A binary presence-absence vector for the first site.
vec_to	A binary presence-absence vector for the second site.

Value

The Phi Coefficient value.

---

plot_ispline_boxplots *Plot facetted boxplots for all ispline basis columns*

Description

Creates a multi-facet boxplot of every ispline basis column in your data, grouped by a specified order factor (e.g. zeta orders). Each spline term (columns ending in _is) gets its own facet, and uses a color-blind-friendly Viridis palette.

Usage

```
plot_ispline_boxplots(
  ispline_data,
  ispline_suffix = "_is",
  order_col = "zOrder",
  palette = "viridis",
  direction = -1,
  ncol = 3,
  outlier_size = 0.5
)
```

Arguments

ispline_data	A data frame as returned by run_ispline_models, containing raw covariates, their spline bases (suffixed _is), and an order column.
ispline_suffix	A string suffix identifying your spline columns. Default is "_is".
order_col	The name of the grouping column (e.g. "zOrder").
palette	One of the Viridis options ("viridis", "magma", "plasma", "cividis", etc.). Default "viridis".
direction	Integer 1 or -1 to control palette direction. Default -1 (reversed).
ncol	Number of columns in the facet wrap. Default 3.
outlier_size	Size of the outlier points. Default 0.5.

Details

1. Automatically detects all columns whose names end with ispline_suffix.
2. Pivots the data to long format for **ggplot2**.
3. Facets a boxplot for each spline variable with independent scales.

Value

A **ggplot2** object showing one boxplot per spline term.

See Also

run_ispline_models, plot_ispline_lines

Examples

```
## Not run:
# load example data
data(bird.spec.fine)
data(bird.env.fine)

# prepare inputs
xy.bird    <- bird.spec.fine[,1:2]
spp.bird   <- bird.spec.fine[,3:102]
envir.bird <- bird.env.fine[,3:9]

# Fit & gather ispline tables
ispline_tabs_all <- run_ispline_models(
  spp_df    = spp.bird,
  env_df    = envir.bird,
  xy_df     = xy.bird,
  orders    = 2:6,
  sam       = 100,
  normalize = "Jaccard",
  reg_type  = "ispline"
)

# Facetted boxplots of all *_is columns
plot_ispline_boxplots(
  ispline_data   = ispline_tabs_all,
  ispline_suffix = "_is",
  order_col      = "zOrder",
  palette        = "viridis",
  direction      = -1,
  ncol           = 3
)

## End(Not run)
```

---

plot_ispline_lines *Plot ispline partial effects with quantile and start-point markers*

Description

Given a tidy data frame of ispline basis outputs (from run_ispline_models), this function identifies the ispline column matching a specified covariate, draws thinner ispline curves for each zOrder, and overlays:

- Small symbols at user-defined quantiles of the raw covariate.
- A larger symbol at each curve's starting point (minimum covariate value).

Usage

```
plot_ispline_lines(
  ispline_data,
  x_var,
  orders = NULL,
  cols = NULL,
  shapes = NULL,
  probs = c(0, 0.25, 0.5, 0.75, 1),
  line_size = 0.5,
  point_size = 1.5,
  start_size = 3,
  start_stroke = 0
)
```

Arguments

ispline_data	A data frame as returned by run_ispline_models, containing raw covariates, their _is spline bases, and a zOrder column.
x_var	A string; the name of the raw covariate to plot (e.g. "dist"). The function will look for a matching spline column named <x_var>_is.
orders	Character vector of zOrder levels (in desired legend/order). Default is unique(ispline_data$zOrder).
cols	Character vector of colours, one per order. Default uses scales::hue_pal().
shapes	Integer vector of plotting symbols (pch codes), one per order. Default is 15:(...).
probs	Numeric vector of probabilities (between 0 and 1) at which to place small quantile markers. Default c(0, .25, .5, .75, 1).
line_size	Numeric; line width for the spline curves. Default 0.5.
point_size	Numeric; size of the quantile markers. Default 1.5.
start_size	Numeric; size of the big start-point markers. Default 3.
start_stroke	Numeric; stroke width for the big start markers. Default 0.

Details

1. Detects all columns ending in _is and fuzzy-matches x_var against them.
2. Extracts the raw covariate name by stripping _is.
3. Computes one "start" point per zOrder at the minimum raw covariate.
4. Computes quantile-closest points per zOrder at the user's probs.

Value

A **ggplot2** object showing:

- `geom_line()` for each curve.
- `geom_point()` at the specified quantiles.
- A larger `geom_point()` at each curve's minimum.

See Also

`run_ispline_models`, `ggplot`

Examples

```
## Not run:
# load sample data
data(bird.spec.fine)
data(bird.env.fine)

# prepare inputs
xy.bird    <- bird.spec.fine[,1:2]
spp.bird   <- bird.spec.fine[,3:102]
envir.bird <- bird.env.fine[,3:9]

# fit & gather ispline tables
ispline_tabs_all <- run_ispline_models(
  spp_df    = spp.bird,
  env_df    = envir.bird,
  xy_df     = xy.bird,
  orders    = 2:6,
  sam       = 100,
  normalize = "Jaccard",
  reg_type  = "ispline"
)

# Line plot for "dist"
plot_ispline_lines(
  ispline_data = ispline_tabs_all,
  x_var        = "dist",            # will match "dist_is"
  orders       = paste("Order", 2:6),
  cols         = c('green','cyan','purple','blue','black'),
  shapes       = c(15,16,17,18,19)
)

## End(Not run)
```

predict_dissim *Predict Pairwise Compositional Turnover (zeta-dissimilarity) with Richness*

Description

Takes raw species and environmental data, fits a multi-site GDM model output, computes predicted pairwise turnover (zeta2) across the landscape, and returns a data frame with site-level richness, environmental covariates, distance, and predicted turnover; optionally plots a heatmap of zeta2 predictions.

Usage

```
predict_dissim(
  grid_spp,
  species_cols,
  env_vars,
  zeta_model,
  grid_xy,
  bndy_fc = NULL,
  x_col = "x",
  y_col = "y",
  show_plot = TRUE,
  skip_scale = FALSE
)
```

Arguments

grid_spp	Data frame containing site IDs, coordinates, and species presence-absence/abundance columns.
species_cols	Integer or character vector giving the columns of grid_spp that hold species data.
env_vars	Data frame of raw environmental predictors (unscaled; rows must align with grid_spp).
zeta_model	Fitted object from Zeta.msgdm (order = 2, reg.type = "ispline").
grid_xy	Data frame of site coordinates, same row-order as grid_spp, with columns x_col, y_col.
bndy_fc	Optional sf or SpatVector polygon to overlay.
x_col	Name of the x (longitude) column in grid_spp/grid_xy.
y_col	Name of the y (latitude) column.
show_plot	Logical; if TRUE (default), print the turnover heatmap.

Value

A data frame with one row per site, containing:

richness Species richness (sum across `species_cols`).

distance Mean great-circle distance (km) from each site to all others.

<env_vars> All scaled environmental predictors.

pred_zeta Linear predictor (logit scale) from `Predict.msgdm()`.

pred_zetaExp Predicted turnover (0-1 scale).

log_pred_zetaExp Natural log of `pred_zetaExp`.

x_col, y_col Site coordinates (from `grid_xy`).

Examples

```
## Not run:
result <- predict_dissim(
  grid_spp     = bird.spec.fine,
  species_cols = 3:102,
  env_vars     = bird.env.fine[,3:9],
  zeta_model   = z_mod,
  grid_xy      = bird.spec.fine[,1:2],
  bndy_fc      = rsa,
  x_col        = "x",
  y_col        = "y",
  show_plot    = FALSE
)

## End(Not run)
```

`richness` *Calculate Species Richness*

Description

Counts unique species per site; reports absolute differences for pairwise and summary stats (range, variance, mean) for multi-site.

Usage

```
richness(vec_from, vec_to = NULL)
```

Arguments

`vec_from`	A numeric vector representing species counts at the first site.
`vec_to`	(Optional) A numeric vector for pairwise or higher-order comparisons.

Value

The species richness value.

`rm_correlated` *Remove Highly Correlated Predictors*

Description

Detects pairs (or groups) of strongly collinear predictors and eliminates the minimum subset necessary to keep every absolute pairwise correlation below a user-defined threshold. Correlations are computed with `stats::cor()`; the variables to discard are chosen via the `caret::findCorrelation()` algorithm. An optional heat-map of the correlation matrix is produced with **corrplot** for rapid inspection.

Usage

```
rm_correlated(data, cols = NULL, threshold = 0.7, plot = TRUE)
```

Arguments

`data`	A data frame with candidate predictor variables.
`cols`	Optional numeric or character vector specifying columns to consider; defaults to **all** numeric columns.
`threshold`	Numeric in $[0, 1]$ specifying the absolute correlation cut-off (default `0.7`).
`plot`	Logical; if `TRUE` (default) a correlation heat-map with coefficients is drawn.

Details

- Non-numeric columns are silently dropped prior to correlation calculation.
- When `cols` is supplied (numeric or character), only those columns are tested; otherwise all numeric columns in `data` are used.
- The names of removed and retained variables are printed to the console for transparency.

Value

A data frame containing the original rows but only the subset of predictor columns whose absolute pairwise correlations are $<$ `threshold`.

See Also

`corrplot`, `findCorrelation`

Examples

```
set.seed(99)
n  <- 200
df <- data.frame(
  a = rnorm(n),
  b = rnorm(n),
  c = rnorm(n) * 0.8 + rnorm(n) * 0.2,  # moderately corr. with 'a'
```

```
  d = rnorm(n) * 0.9 + rnorm(n) * 0.1   # moderately corr. with 'b'
)

## Remove predictors with r >= 0.75
df_reduced <- rm_correlated(df, threshold = 0.75, plot = FALSE)
names(df_reduced)

## Visualise the correlation structure & removals at a stricter threshold
rm_correlated(df, threshold = 0.6, plot = TRUE)
```

`run_ispline_models` *Run multiple Zeta.msgdm ispline models and return both models and combined ispline table*

Description

Fits `Zeta.msgdm` models of type "ispline" for a series of zeta-orders, extracts the raw environmental covariates (plus distance) and their ispline bases, and returns both the list of fitted models and one tidy data frame combining all orders.

Usage

```
run_ispline_models(
  spp_df,
  env_df,
  xy_df,
  orders = 2:6,
  sam = 100,
  distance.type = "Euclidean",
  normalize = "Jaccard",
  reg_type = "ispline"
)
```

Arguments

`spp_df`	A data frame or matrix of species incidence/abundance.
`env_df`	A data frame of environmental covariates.
`xy_df`	A two-column data frame or matrix of site coordinates.
`orders`	Integer vector of zeta orders to fit (e.g. 2:6).
`sam`	Integer; number of random samples per order (passed to `Zeta.msgdm`).
`distance.type`	Character; distance metric for `Zeta.msgdm` (default "Euclidean").
`normalize`	Character; normalization method for `Zeta.msgdm` (default "Jaccard").
`reg_type`	Character; regression type for `Zeta.msgdm` (default "ispline").

Value

A named list with:

`zeta_gdm_list` A list of the fitted `Zeta.msgdm()` objects, named by order.

`ispline_table` A tibble with one row per sample, containing all original covariates (including `distance`), the ispline bases suffixed `_is`, and a `zOrder` column.

See Also

`Zeta.msgdm`, `Return.ispline`

Examples

```
## Not run:
data(bird.spec.fine); data(bird.env.fine)
xy   <- bird.spec.fine[,1:2]
spp  <- bird.spec.fine[,3:102]
env  <- bird.env.fine[,3:9]

out <- run_ispline_models(
  spp_df        = spp,
  env_df        = env,
  xy_df         = xy,
  orders        = 2:6,
  sam           = 100,
  normalize     = "Jaccard",
  reg_type      = "ispline"
)
names(out)
head(out$ispline_table)

## End(Not run)
```

---

turnover *Calculate Species Turnover or Beta Diversity*

Description

Computes beta diversity as proportion unshared; 0-1 for pairwise, extended to multi-site averages or totals.

Usage

```
turnover(vec_from, vec_to = NULL)
```

Arguments

`vec_from`	A numeric vector representing species counts at the first site.
`vec_to`	(Optional) A numeric vector for pairwise or higher-order comparisons.

Value

The species turnover value.

Index