Package: natto (via r-universe)

August 27, 2024

Title An Extreme 'vegan' Package of Experimental Code

Version 0.3

Description Random code that is too experimental or too weird to be included in the vegan package.

License MIT + file LICENCE

Imports vegan (>= 2.6.3), cluster

Suggests knitr, dendextend, MASS

VignetteBuilder knitr

URL https://github.com/jarioksa/natto

LazyData true

Encoding UTF-8

RoxygenNote 7.3.1

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

RemoteUrl https://github.com/jarioksa/natto

RemoteRef HEAD

RemoteSha 7a5b04d0edda38ee8d490589e22d9bae32991214

Contents

as.dist.deldir — *Cast deldir Object to Distances Between Neighbours*

Description

Function casts a Delaunay triangulation from `deldir` (**deldir** package) to a `dist` object, with numeric values for distances between neighbours and `NA` for other distances.

Usage

```
## S3 method for class 'deldir'
as.dist(m, diag = FALSE, upper = FALSE)
```

Arguments

`m`	A `deldir` result.
`diag`	logical value indicating whether the diagonal of the distance matrix should be printed by `print.dist`.
`upper`	logical value indicating whether the upper triangle of the distance matrix should be printed by `print.dist`.

`as.prcomp` *Cast vegan::rda Result to base::prcomp*

Description

Function casts a result object of unconstrained `rda` to a `prcomp` result object.

Usage

```
as.prcomp(x)

## S3 method for class 'rda'
as.prcomp(x)
```

Arguments

`x`	An unconstrained `rda` result object.

`bayesjaccard` *Community Dissimilarity as Expected or Sampled Beta Variate*

Description

Jaccard dissimilarity for presence/absence data can be seen as the mode of Beta distributed variate. The function calculates the dissimilarity as a random sample of Beta distribution, or alternatively as its expected value or mode.

Usage

```
bayesjaccard(x, method = c("rbeta", "expected", "mode"))
```

Arguments

`x`	Community data, will be treated as binary.
`method`	Dissimilarity as a random sample from Beta distribution or as its expected value or mode.

Details

In often-used 2x2 contingency table notation, a is the number of species shared by two compared communities, and b and c are the numbers of species occurring only in one of the compared communities. Assuming uniform prior in (0, 1), the species will “sample” the dissimilarity of two communities with posterior Beta($b + c + 1$, $a + 1$). This will have mode $(b + c)/(a + b + c)$ and expected value $(b+c+1)/(a+b+c+2)$. The mode is directly the Jaccard dissimilarity for binary (presence/absence) data, and the expected value adds 1 in numerator and 2 in denominator from the prior. The importance of prior will diminish when the number of species $a + b + c$ grows, but it will protect from claiming complete identity or complete difference when we only have a few species.

Function `bayesjaccard` estimates Jaccard dissimilarity as Beta-distributed random variate, and will return random sample from its posterior distribution. It can also return the expected value or the mode which are constant in function calls.

The function is intended to be used to assess the effect of random sampling variation in community analysis. The **natto** package provided three examples of its application: `clsupport` finds the "support" of branches in hierarchic clustering by function `hclust`, `bjNMDS` the variation of ordination scores in non-metric multidimensional scaling by functions `metaMDS` and `monoMDS`, and `bjdbrda` the variation of ordination scores of constrained component of distance-based RDA by function `dbrda`. All these functions find the basic result from the expected value of the dissimilarity, and produce a set of random samples from the Beta distribution to assess the variation in the result.

Value

Dissimilarity object inheriting from classes `"dist"` and `"designdist"`.

See Also

Function is a wrapper to `designdist`.

Examples

```
data(spurn)
## the effect of prior
plot(bayesjaccard(spurn, "mode"), bayesjaccard(spurn, "expected"))
abline(0, 1, col=2)
## one random sample of dissimilarities
plot(bayesjaccard(spurn, "expected"), bayesjaccard(spurn), asp=1)
abline(0, 1, col=2)
```

`BCI.env2` *Habitat Types of Barro Colorado Island Forest Plots*

Description

Habitat types of Barro Colorado Island Forest Dynamics Plots classified in 25 grid cells in each 1-ha plot (Harms et al. 2001).

Usage

```
data("BCI.env2")
```

Format

A data frame with 50 observations on the area (in ha) of following habitat types:

`Young` Young forests of ca. 100 years.

`Stream` Streamside vegetation.

`Swamp` Swamp.

`OldHigh` Old forests on flat surface above 152 m altitude.

`OldLow` Old forests on flat surface at or below 152 m altitude.

`OldSlope` Old forests on steep slopes (> 7 degrees).

`Mixed` Mixed habitat.

Details

The data set provides habitat type classification for the Barro Colorado Island data `BCI` in the **vegan** package. The **vegan** package provides alternative environmental data `BCI.env` which has variables derived from this data set: Dominant habitat type excluding `Stream`, presence of `Stream` grid cells in the plot, and habitat type diversity of each plot (see Examples).

The data were extracted from Harms et al. (2001: Fig. 1) who give a map of 50 forest plots divided into 25 grid cells, and the areas are expressed in ha (at 0.04 ha resolution).

Source

Harms et al. (2001), Fig. 1.

References

Harms K.E., Condit R., Hubbell S.P. & Foster R.B. (2001) Habitat associations of trees and shrubs in a 50-ha neotropical forest plot. *J. Ecol.* 89, 947–959.

Examples

```
## Derive variables in BCI.env data in vegan
data(BCI.env2)
## 1. The dominant habitat type in each plot
k <- apply(BCI.env2[,-2], 1, which.max)
Habitat <- (names(BCI.env2)[-2])[k]
## 2. Presence of streamside habitats in the plot
Stream <- with(BCI.env2, ifelse(Stream > 0, "Yes", "No"))
## 3. Habitat type diversity (needs vegan)
if (require(vegan)) {
EnvHet <- diversity(BCI.env2, "simpson")
}

## The combined variables in BCI.env and the original area variables in
##  BCI.env2 give nearly equal constraints.
if (require(vegan)) {
data(BCI)
## habitat areas per 1-ha plot
ord <- cca(BCI ~ ., BCI.env2)
## dominant habitat type of each 1-ha plot
ord0 <- cca(BCI ~ Habitat)
plot(procrustes(ord, ord0))
}
```

BCI.taxon *Botanical Classification of Barro Colorado Island Species*

Description

Classification of `BCI` species based on APG IV (2016) and Chase & Reveal (2009).

Usage

```
data("BCI.taxon")
```

Format

A data frame with 225 observations on the following 7 variables.

`genus` Genus.

`family` APG IV family.

`order` APG IV order.

`superorder` Superorder (Chase & Reveal, 2009).

`unrank` Either `eudicotyledons`, `Magnoliidae` or `monocotyledons` which correspond to traditional major clades, but have no formal status in the classification of the subclass of Angiosperms.

References

APG IV [Angiosperm Phylogeny Group] (2016) An update of the Angiosperm Phylogeny Group classification for the orders and families of flowering plants: APG IV. *Bot. J. Linnean Soc.* **181**: 1–20.

Chase, M.W. & Reveal, J. L. (2009) A phylogenetic classification of the land plants to accompany APG III. *Bot. J. Linnean Soc.* **161**: 122–127.

Examples

```
data(BCI.taxon)
```

bjdbrda *Multiple dbRDA from Beta Distributed Jaccard Dissimilarity*

Description

Function performs distance-based RDA on expected Beta Jaccard dissimilarities, and then reruns the analysis on given number of random Beta Jaccard dissimilarities.

Usage

```
bjdbrda(formula, data, n = 100, ...)

## S3 method for class 'bjdbrda'
scores(
  x,
  choices = 1:2,
  display = c("wa", "lc", "bp", "cn"),
  scaling = "species",
  const,
  expected = TRUE,
  ...
)

## S3 method for class 'bjdbrda'
plot(
  x,
  choices = 1:2,
  wa = "p",
  lc = "n",
  cn = "hull",
  bp = "wedge",
  wa.par = list(),
  lc.par = list(),
  cn.par = list(),
  bp.par = list(),
  scaling = "species",
  type = "t",
  ...
)

## S3 method for class 'bjdbrda'
boxplot(
  x,
  kind = c("eigen", "correlation"),
  points = "red",
  pch = 16,
  xlab = "Axis",
  ylab,
  ...
)
```

Arguments

`formula, data`	Model definition of type `Y ~ Var1 + Var2,data = X`, where `Y` is dependent community data (handled as binary data), `Var1` and `Var2` are independent (explanatory) variables found in data frame `X`. See `dbrda` for further information.
`n`	Number of Jaccard dissimilarity matrices sampled from Beta distribution.
`...`	Other parameters passed to functions; passed `dbrda` in `bjdbrda`, ignored in `scores`.
`x`	`bjdbrda` result object.
`choices`	Selected ordination axes.
`display, scaling, const`	Kind of scores, the scaling of scores (axes), and scaling constant with similar definitions as in `scores.rda`.
`expected`	Return scores of the expected ordination instead of ordination based on random samples from Jaccard dissimilarity.
`wa, lc, cn, bp`	Display of corresponding scores. `"n"` skips the score, `"p"` and `"t"` use points or text for the expected score, and other shapes define `bjpolygon` or `bjstars` shape used for sampled random scores.
`wa.par, lc.par, cn.par, bp.par`	List of arguments to modify the plotting parameters of the corresponding shape.
`type`	Add `"t"`ext or `"p"`oint for the expected score to a shape of scatter of random shapes.
`kind`	Draw boxplots of eigenvalues or pairwise axis correlations for each random sample.
`points, pch`	Add points of given color and shape to the eigenvalue boxplot.
`xlab, ylab`	Change labelling of axes in boxplot.

Details

Function works only with constrained and partial constrained ordination, and only saves the randomized results of constrained ordination. To maintain the eigenvalues, function does not rotate the random results to the reference ordination, but it will reflect axes with reversed signs. The eigenvalues and magnitudes of axiswise correlations to the reference ordination can be inspected with `boxplot`.

The display type in `plot` can be specified independently to each type of score (or the score can be skipped). The default display can be modified using a similarly named list of graphical argument values that will replace the defaults. For instance, to display linear combination scores as convex hull, use `lc = "hull"`, and modify its parameters with list `lc.par`. For available graphical parameters, see `bjpolygon` for filled shapes, `bjstars` for stars, `points` for points, and `ordilabel` for text. Alternatives `"p"` and `"t"` will only show the scores of the reference solution.

Value

Function returns dbrda result object amended with item `BayesJaccard` that is a list of randomized results with following items for each random sample:

- `tot.chi` total inertia
- `eig` eigenvalues of constrained axes
- `r` absolute value of correlation coefficient of randomized axis and the reference axis
- `u, wa, biplot, centroids` ordination scores for linear combination scores, weighted averages scores, biplot scores and centroid of factor constraints; these are unscaled and are usually accessed with `scores.bjdbrda` for appropriate scaling

Examples

```
if (require(vegan)) {
data(dune, dune.env)
m <- bjdbrda(dune ~ A1 + Management + Moisture, dune.env)
## eigenvalues and correlations showing axis stability
boxplot(m)
boxplot(m, kind = "correlation")
## Default plot
plot(m)
## modify plot
plot(m, cn = "star", wa = "ellipse", cn.par = list(col=gl(2,4)),
   wa.par=list(col = dune.env$Management, keep = 0.607))
}
```

`bjNMDS` *Multiple NMDS Ordinations from Beta Distributed Jaccard Dissimilarity*

Description

Function performs `metaMDS` on expected Beta Jaccard dissimilarity with multiple starts and then a number of `monoMDS` runs on random Beta Jaccard dissimilarities starting from the expected configuration, and Procrustes rotates the random solutions to the expected one.

Usage

```
bjNMDS(
  x,
  n = 100,
  trymax = 500,
  maxit = 1000,
  smin = 1e-04,
  sfgrmin = 1e-07,
  sratmax = 0.999999,
  parallel = 2,
  trace = FALSE,
  ...
)

## S3 method for class 'bjnmds'
plot(
  x,
  choices = 1:2,
  kind = c("hull", "ellipse", "wedge", "star"),
  keep = 0.9,
  type = "t",
  ...
)
```

Arguments

`x`	Community data to be analysed and treated as binary (`bjNMDS`) or object to be plotted (`plot`).
`n`	Number of random samples of Beta Distribution.
`trymax`	Maximum number of random starts in `metaMDS`.
`maxit, smin, sfgrmin, sratmax`	Convergence parameters in `monoMDS`.
`parallel`	Number of parallel tries in `metaMDS`.
`trace`	Trace iterations in `metaMDS`.
`...`	Other parameters passed to functions.
`choices`	Axes to be plotted.
`kind`	Shape to be plotted to show the scatter of coordinates of sampled distances; see `bjpolygon` for details.
`keep`	Proportion of points to be enclosed by shape; see `peelhull` and `peelellipse`.
`type`	Type of the plot: `"t"`ext, `"p"`oints or `"n"`one.

Details

Current function is designed for visual inspection of random variation of ordination, and no numerical analysis functions are (yet) available. The `plot` can show the scatter of points as convex hulls or ellipsoid hulls containing a given proportion of random points, or as stars that connect the expected point to randomized ones. With option `"wedge"` the convex hull is extended to include the observed point if necessary. See `bjpolygon` and `bjstars` for technical details.

Missing functionality includes adding species scores, adding fitted environmental vectors and factors and numeric summaries of the randomization results.

Value

A `metaMDS` result object amended with item `rscores` that is a three-dimensional array of coordinates from random `bayesjaccard` ordinations.

Examples

```
data(spurn)
m <- bjNMDS(spurn)
plot(m, keep = 0.607, col = "skyblue")
```

`bjpolygon` *Shapes to Display Scatter of Points in Multiple Ordinations*

Description

These are low-level functions that are used by `plot` functions to draw shapes enclosing a specified proportion of randomized points, or connecting certain proportion of them to the reference points. The functions are not usually called directly by users, but the plots can be modified with described arguments.

Usage

```
bjpolygon(
  xarr,
  x0,
  keep = 0.9,
  kind = c("hull", "ellipse"),
  criterion = "area",
  linetopoint = TRUE,
  col = "gray",
  alpha = 0.3,
  observed = TRUE,
  type = c("t", "p", "n"),
  ...
)

bjstars(xarr, x0, keep = 0.9, col = "gray", type = c("t", "p", "n"), ...)
```

Arguments

`xarr`	3-D array of coordinates of sampling units times two axes by random samples.
`x0`	2-D array of sampling units times two axes treated as constant for all samples in `xarr`.
`keep`	Proportion of points enclosed in the shape; passed to `peelhull` or `peelellipse`.
`kind`	Shape is either a convex hull (`peelhull`) or ellipse `peelellipse` enclosing keep proportion of `xarr` points.
`criterion`	Criterion to remove extreme points from the hull in `peelhull` and `peelellipse`.
`linetopoint`	Draw line from the centre of the shape to the coordinates in `x0`.
`col`	Colour of the shape; can be a vector of colours.
`alpha`	Transparency of shapes; 0 is completely transparent, and 1 is non-transparent.

`observed`	After finding the convex hull, extend hull to enclose fixed point `x0`.
`type`	Mark coordinate of `x0` using `"t"`ext, `"p"`oint or `"n"`one.
`...`	Other parameters passed to to marker of `type`.

Details

Functions require output of three-dimensional array `xarr` of randomized scores, where the last dimension is for the random samples, and a two-dimensional matrix of references scores `x0`.

Function `bjpolygon` uses `peelhull` or `peelellipse` to find a convex hull (`chull`) or an ellipsoid hull (`ellipsoidhull`) containing `keep` proportion of points. The reference coordinates can also be added to the plot, either as point or a text label (`ordiellipse`) and optionally connected to the shape centre (not to the centre of points). With argument `observed` the convex hull can be extended to include the reference point when that is outside the hull; this is used to draw wedges for arrows using the origin as the reference point.

Function `bjstars` connects the reference point to `keep` proportion of closest points. If the reference point is not specified, the centre of points prior to selection will be used.

Value

Functions return invisibly `NULL`.

See Also

`peelhull`, `peelellipse`, `polygon` for basic functions and `plot.bjnmds` and `plot.bjnmds` and `plot.bjdbrda` for programmatic interface.

`canneddist` *Canned Dissimilarities with their Vernacular Names*

Description

Function is a storehouse for dissimilarity indices that can be called by their vernacular names. The function is based on `designdist` (**vegan** package).

Usage

```
canneddist(x, method, help = FALSE)
```

Arguments

`x`	Input data.
`method`	Vernacular name for a dissimilarity index.
`help`	List available indices and their definitions instead of calculating dissimilarities.

Details

Function wraps popular dissimilarity indices for `designdist` allowing these to be called by their popular names. It can have synonymous names for one dissimilarity index. Use argument `help=TRUE` to see the current selection of indices and their definitions.

The function uses the main notation of `designdist` where terms are based on sums and paired minima or sum of squares and crossproducts of pairs of sampling units. For vectors `x` and `y` the `"quadratic"` terms are `J = sum(x*y)`, `A = sum(x^2)`, `B = sum(y^2)`, and `"minimum"` terms are `J = sum(pmin(x,y))`, `A = sum(x)` and `B = sum(y)`, and `"binary"` terms are either of these after transforming data into binary form (number of shared species `J`, and number of species for each row, `A`, `B`.). Number of columns (species) is notated as `P`, and the number of sampling units is `N`.

There is a huge number of indices, and the current selection is far from comprehensive (but it can easily expanded). See References for sources. Many sources use different notation, but they were changed to the notation described above. For instance, in popular (but strange) 2x2 contingency table notation for binary data `a = J`, `b = A-J`, `c = B-J`, `d = P-A-B+J`. Some of formulae may be surprising, but they are mathematically equivalent to traditional ones. I challenge you to inspect Euclidean distance, and once you see how it is derived, try Chord distance.

Value

Function returns a `dist` object of dissimilarities.

Author(s)

Jari Oksanen

References

Hubálek, Z. (1982). Coefficients of association and similarity, based on binary (presence-absence) data: an evaluation. *Biological Review* 57, 669–689.

Legendre, P. & Legendre, L. (2012). *Numerical Ecology.* 3rd English Ed., Elsevier.

Yue, J.C. & Clayton, M.K. (2005). A similarity measure based on species proportions. *Communications in Statistics Theory and Methods* 34, 2123–2131. doi:10.1080/STA200066418.

See Also

Function is a wrapper to `designdist`. **vegan** function `betadiver` is a similar collection of canned indices for beta diversity, and many of these are well-known dissimilarity indices.

Examples

```
data(spurn)
## Ochiai dissimilarity
canneddist(spurn, "ochiai")
```

`clsupport` *Support of Branches of Hierarchical Clustering from Beta Jaccard*

Description

Function performs hierarchic clustering (`hclust`) with the expected value of Jaccard dissimilarity as Beta random variate, and compares the stability of each cluster branch in random samples from Beta distribution.

Usage

```
clsupport(x, n = 1000, method = "average", softmatch = FALSE, plot = TRUE, ...)

clsets(hclus)
```

Arguments

`x`	Community data, will be treated as binary.
`n`	Number of random samples from Beta distribution.
`method`	Clustering method, passed to `hclust`.
`softmatch`	Estimate cluster similarity as Jaccard similarity or as a hard exact match between two clusters.
`plot`	Plot the `hclust` dendrogram from expected Jaccard dissimilarity with each brand labelled with support.
`...`	Other parameters; passed to `plot.hclust`.
`hclus`	`hclust` result object.

Details

Function is basically a graphical tool that plots an `hclust` dendrogram and adds the count of similar clusters in randomized cluster dendrograms, but it can also be called only for the numeric result without plot.

The count of similar clusters in trees is based on randomized form Jaccard Beta dissimilarities, and is called here “support”. The support is calculated alternatively as the number of exactly similar branches in randomized trees or as a sum of maximum Jaccard similarities in observed and randomized trees (“softmatch”). The exact matches are very sensitive to single wandering sampling units, and often give very low “support” for large classes, whereas Jaccard-based “support” may give too high values for the smallest classes. For exact matches the “support” is given as count, and for soft maches (Jaccard-based) as a rounded integer per 1000.

Value

Usually called to draw a plot, but will return the “support”; the values are returned in the order of merges in the agglomerative hierarchic clustering. Function `clsets` returns a list with indices of items (sampling units) of each branch in their merge order.

See Also

hclust, bayesjaccard.

Examples

```
data(spurn)
clsupport(spurn) # exact match
clsupport(spurn, softmatch = TRUE)
```

---

distconstrain *Constrained and Residual Dissimilarities*

Description

Function constrains dissimilarities by external variables, or alternatively removes effects of constraining variables and returns residual dissimilarities. The analysis is based on McArdle & Anderson (2001), and the analysis of constrained dissimilarities is equal to distance-based Redundancy Analysis (dbrda).

Usage

```
distconstrain(formula, data, add = FALSE, residuals = FALSE, squared = FALSE)
```

Arguments

formula	The left-hand-side must be dissimilarities and the right-hand-side should list the constraining variables.
data	Data frame containing the constrainging variables in the `formula`.
add	an additive constant is added to the non-diagonal dissimilarities such that all $n - 1$ eigenvalues are non-negative. Alternatives are `"lingoes"` (default, also used with `TRUE`) and `"cailliez"` (which is the only alternative in cmdscale).
residuals	Return residuals after constraints.
squared	Return squared dissimilarities instead of dissimilarities. This allows handling negative squared distances by the user instead of setting them `NaN`.

Details

Function uses the method of McArdle & Anderson (2001) to constrain dissimilarities by external variables, or alternatively, to find residual dissimilarities after constraints. With Euclidean distances, the method is equal to performing linear regressions on each column in the raw data and then calculationg the distances, but works directly on distances. With other methods, there is no similar direct connection to the raw data, but it is possible to work with non-Euclidean metrics. The same basic method is used within db-RDA (dbrda in **vegan**), but this function exposes the internal calculations to users.

Non-Euclidean indices can produce negative eigenvalues in db-RDA. Would negative eigenvalues be produced, this function can return negative squared distances resulting in `NaN` when taking the square root. Db-RDA works with the internal presentation of the dissimilarities, and its analysis does not suffer from the imaginary distances, but these can ruin the analysis of dissimilarities returned from this function.

References

McArdle, B.H. & Anderson, M.J. (2001). Fitting multivariate models to community data: a comment on distance-based redundancy analysis. *Ecology* 82, 290–297.

`distMeans` *Mean of Distances*

Description

Mean of distances is defined as the distance of each point to the mean or expected value of coordinates generating the distances.

Usage

```
distMeans(...)

## Default S3 method:
distMeans(d, addcentre = FALSE, label = "centroid", ...)

## S3 method for class 'formula'
distMeans(formula, data, ...)
```

Arguments

`...`	Other parameters (ignored).
`d`	Distances as a `dist` object
`addcentre`	Add distances to the centroid as the first item in the distance matrix. If `FALSE` only return mean distances.
`label`	Label for the centroid when `addcentre` = `TRUE`.
`formula, data`	Formula where the left-hand-side is the dissimilarity structure, and right-hand-side defines the mean from which the dissimilarities are calculated. The terms in the right-hand-side can be given in `data`.

Details

Function is analagous to `colMeans` or `rowMeans` and returns values that are at the mean of distances of each row or column of a symmetric distance matrix. Alternatively, the use of formula calculates mean distances to the fitted values.

Means of distances cannot be directly found as marginal means of distance matrix, but they must be found after Gower double centring (Gower 1966). After double centring, the means are zero, and when backtransformed to original distances, these give the mean distances. When added to the original distances, the metric properties are preserved. For instance, adding centres to distances will not influence results of metric scaling, or the rank of spatial Euclidean distances. The method is based on Euclidean geometry, but also works for non-Euclidean dissimilarities. However, the means of very strongly non-Euclidean indices may be imaginary, and given as `NaN`.

Average mean distances can be regarded as a measure of beta diversity, and formula interface allows analysis of beta diversity within factor levels or with covariates. Such analysis is preferable to conventional averaging of dissimilarities or regression analysis of dissimilarities. Analysis of mean distances is consistent with directly grouping observed rectangular data, and overall beta diversity can be decomposed into components defined by the formula, and handles inflating n observations to $n(n-1)/2$ dissimilarities in analysis.

Value

Distances to all other points from a point that is in the centroid or fitted value (with formula interface) of the coordinates generating the distances. Default method allows returning the input dissimilarity matrix where the mean distances are added as the first observation.

Author(s)

Jari Oksanen.

References

Gower, J.C. (1966) Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika* **53**, 325-328.

Examples

```
## Euclidean distances to the mean of coordinates ...
xy <- matrix(runif(5*2), 5, 2)
dist(rbind(xy, "mean" = colMeans(xy)))
## ... are equal to distMeans ...
distMeans(dist(xy))
## ... but different from mean of distances
colMeans(as.matrix(dist(xy)))
## adding mean distance does not influence PCoA of non-Euclidean
## distances (or other metric properties)
data(spurn)
d <- canneddist(spurn, "bray")
m0 <- cmdscale(d, eig = TRUE)
mcent <- cmdscale(distMeans(d, addcentre=TRUE), eig = TRUE)
## same non-zero eigenvalues
zapsmall(m0$eig)
zapsmall(mcent$eig)
## distMeans are at the origin of ordination
head(mcent$points)
```

diverclust *beta and gamma Diversity Clustering*

Description

Function forms hierarchic clustering so that each fusion minimizes beta deiversity or gamma diversity.

Usage

```
diverclust(
  x,
  renyi = 1,
  equalize = TRUE,
  beta = TRUE,
  hill = FALSE,
  trace = TRUE
)
```

Arguments

`x`	Community data.
`renyi`	The scale of Renyi diversity index as defined in `renyi`. Value `0` gives diversity based on species richness, `1` diversity based on Shannon entropy, `2` diversity based on Simpson index, and `Inf` diversity based on Berger-Parker index. Other non-negative values are also allowed, but they may not define a well known standard diversity index.
`equalize`	Equalize data rows so that they can be meaningfully pooled and averaged. If this is `FALSE`, raw data will be used, and beta diversities may be negative. The equalization depends on the value of `renyi`, and each row of `x` is divided by `(rowSums(x^renyi))^(1/renyi)`. In borderline cases `renyi=0` the data are presence-absence tranformed (but can be left untransformed), and with `renyi=Inf` the rows are divided by row maxima (see `decostand`).
`beta`	Use beta diversities: the average alpha diversity of cluster members is subtracted from the pooled diversity. If this is `FALSE`,the clustering is based on pooled diversity, also known as gamma diversity.
`hill`	Use Hill numbers instead of Renyi diversity (see `renyi`). For `renyi = 0` these are species richess values instead of their logarithms, and for `renyi = 1` they are exponents of Shannon diversity instead of Shannon diversities. In general, a Hill number is an exponent of Renyi diversity. The Hill numbers may not be strictly additive, and beta diversities may be negative (except with `renyi = 0`).
`trace`	Trace calculations. Either logical or an integer: `trace=2` also traces merges.

Details

The function forms clusters so that pooled diversity or its change to the baseline alpha diversity is minimized at each level of clustering. The change of pooled diversity with respect to the baseline alpha diversity is called beta diversity, and the overall diversity is called gamma diversity. For beta diversity, the clustering implies an additive partitioning.

Value

Function returns an `hclust` object.

Author(s)

Jari Oksanen.

See Also

`hclust` for cluster analysis and its support methods, `renyi` for estimating Renyi diversities and Hill numbers, and `adipart` for related additive partitioning of beta diversity.

`diverdist` *beta and gamma Diversity Distance*

Description

The distance between two sampling units is the increase in diversity when these are merged together, or the beta diversity between two SUs. Alternatively, the function can find the total diversity or distance from the origin which is known as gamma diversity for two SUs. Any Renyi diversity can be used.

Usage

```
diverdist(x, renyi = 1, equalize = TRUE, beta = TRUE, hill = FALSE)
```

Arguments

`x`	Community data.
`renyi`	The scale of Renyi diversity index as defined in `renyi`. Value `0` gives diversity based on species richness, `1` diversity based on Shannon entropy, `2` diversity based on Simpson index, and `Inf` diversity based on Berger-Parker index. Other non-negative values are also allowed, but they may not define a well known standard diversity index.
`equalize`	Equalize data rows so that they can be meaningfully pooled and averaged. If this is `FALSE`, raw data will be used, and beta diversities may be negative. The equalization depends on the value of `renyi`, and each row of `x` is divided by `(rowSums(x^renyi))^(1/renyi)`. In borderline cases `renyi=0` the data are presence-absence tranformed (but can be left untransformed), and with `renyi=Inf` the rows are divided by row maxima (see `decostand`).

`beta`	Use beta diversities: the average alpha diversity of cluster members is subtracted from the pooled diversity. If this is `FALSE`,the clustering is based on pooled diversity, also known as gamma diversity.
`hill`	Use Hill numbers instead of Renyi diversity (see `renyi`). For `renyi = 0` these are species richess values instead of their logarithms, and for `renyi = 1` they are exponents of Shannon diversity instead of Shannon diversities. In general, a Hill number is an exponent of Renyi diversity. The Hill numbers may not be strictly additive, and beta diversities may be negative (except with `renyi = 0`).

Value

A dissimilarity object inheriting from `dist`.

Author(s)

Jari Oksanen

`ginkgogram` *Plot Dendrogram with Leaves as Fans*

Description

Function is similar to standard `hclust`, but the leaves are drawn as fans with base proportional to weights (or sizes) of leaves.

Usage

```
ginkgogram(
  x,
  labels = NULL,
  check = TRUE,
  axes = TRUE,
  frame.plot = FALSE,
  ann = TRUE,
  main = "Cluster Ginkgogram",
  sub = NULL,
  xlab = NULL,
  ylab = "Height",
  w,
  col,
  leaf = Inf,
  ...
)
```

Arguments

`x`	An object of the type produced by `hclust`.
`labels`	A character vector of labels for te leaves of the tree.
`check`	Logical indicating if the `x` should be checked for validity.
`axes, frame.plot, ann`	Logical flags as in `plot.default`.
`main, sub, xlab, ylab`	Character strings to replace default annotation.
`w`	Weights of leaves.
`col`	Colour of the fill of leaves.
`leaf`	Maximum height of leaf triangle. Leaf height is smaller of this value and the height of the branch, and the default value (`Inf`) always draws leaves to the root. Special value `TRUE` will take the smallest branch height as the maximum height of each leaf.
`...`	Further graphical arguments

Details

Function `ginkgogram` has two new arguments to `hclust`: `w` for weights that give the base width of leaves, and `col` that gives the colour of the fill of leaves. The weights `w` must be numeric and non-negative, and zero weight guarantees that the leaf is drawn as a simple line similarly as in `hclust`.

Examples

```
## need vegan data sets
if (require(vegan)) {
data(dune, dune.phylodis, dune.taxon)
cl <- hclust(dune.phylodis)
ginkgogram(cl, w = colMeans(dune), col = factor(dune.taxon$Superorder))
## limit heights of the polygons to the Cretaceous
ginkgogram(cl, w = colMeans(dune), col = factor(dune.taxon$Superorder),
    leaf = 65)
}
```

`gnlmod` *Generalized Non-Linear Regression Analysis*

Description

Non-linear regression (`nls`) `selfStart` model generalized to use exponential `family` error distributions.

Usage

```
gnlmod(formula, data, family = gaussian, wts, ...)

## S3 method for class 'gnlmod'
predict(object, newdata, type = "response", ...)

## S3 method for class 'gnlmod'
summary(object, dispersion = NULL, correlation = FALSE, ...)
```

Arguments

`formula`	Model formula for a `selfStart` `nls` model.
`data`	Data for formula.
`family`	Error distribution of the exponential `family`.
`wts`	Prior weights
`...`	Other parameters passed to `nlm` that performs the actual regression.
`object`	`gnlmod` result object.
`newdata`	New independent data for prediction.
`type`	Type of predicted values; only `"response"` implemented.
`dispersion`	The dispersion parameter for the family used. Either a single numerical value of `NULL` (the default), when it is inferred from the `object`.
`correlation`	logical; if `TRUE`, the correlation matrix of the estimated parameters is returned and printed.

Details

Function combines three R functions for fitting non-linear models with exponential family errors:

1. Non-linear regression is defined by `selfStart` like in `nls`. In addition to defining the nonlinear function, these also provide initial values and derivatives of model parameters.
2. `family` functions are used to define the error distribution allowing use of other than least squares models. The `family` functions provide the log-likelihood function for fitting, and allows finding the partial derivatives of model parameters for the log-likelihood function (see McCullagh & Nelder 1989, p. 41).
3. The log-likelihood function (with partial derivatives) is optimized with `nlm`.

The result is mostly similar to `glm` object, and can be handled with many `glm` method functions, except those that assume linear fit.

Value

Function returns the `nlm` result object, but it is amended with `glm` object related to `family`.

References

McCullagh, P & Nelder, J.A. (1989) *Generalized Linear Models*, 2nd ed. Chapman & Hall.

Examples

```
## Compare to vegan species-area models
require(vegan) || stop("needs vegan")
data(sipoo, sipoo.map)
S <- specnumber(sipoo)
## Arrhenius model in vegan with least squares
nls(S ~ SSarrhenius(area, k, z), sipoo.map)
## Assume Poisson error
(mod <- gnlmod(S ~ SSarrhenius(area, k, z), sipoo.map, family=poisson))
## Arrhenius should be identical to Poisson glm with log-link
glm(S ~ log(area), sipoo.map, family=poisson) # (Intercept) = log(k)
## some method functions
fitted(mod)
predict(mod)
predict(mod, newdata = list(area = seq(1, 250, len=31)))
summary(mod)
```

---

gradrescale *Rescale Ecological Gradient to Constant Community Heterogeneity*

Description

Rescale Ecological Gradient to Constant Community Heterogeneity

Usage

```
gradrescale(grad, comm)
```

Arguments

grad	Observed numeric gradient.
comm	Community data matrix.

Details

Observed environmental gradient is rescaled similarly as axis in rdecorana ahd decorana. The goal is that weighted averages of species have equal dispersion along the rescaled gradient.

Value

Rescaled gradient scaled in units of heterogeneity.

humpfit *No-interaction Model for Hump-backed Species Richness vs. Biomass*

Description

Function humpfit fits a no-interaction model for species richness vs. biomass data (Oksanen 1996). This is a null model that produces a hump-backed response as an artifact of plant size and density.

Usage

```
humpfit(mass, spno, family = poisson, start)

## S3 method for class 'humpfit'
lines(x, segments = 101, ...)

## S3 method for class 'humpfit'
plot(
  x,
  xlab = "Biomass",
  ylab = "Species Richness",
  lwd = 2,
  l.col = "blue",
  p.col = 1,
  type = "b",
  ...
)

## S3 method for class 'humpfit'
points(x, ...)

## S3 method for class 'humpfit'
predict(object, newdata = NULL, ...)

## S3 method for class 'humpfit'
profile(fitted, parm = 1:3, alpha = 0.01, maxsteps = 20, del = zmax/5, ...)

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

Arguments

mass	Biomass.
spno	Species richness.
family	Family of error distribution. Any family can be used, but the link function is always Fisher's diversity model, and other link functions are silently ignored.
start	Vector of starting values for all three parameters.

`x`	Fitted result object.
`segments`	Number of segments used for lines.
`...`	Other parameters to functions.
`xlab`, `ylab`	Axis labels.
`lwd`	Line width.
`l.col`, `p.col`	Line and point colours.
`type`	Type of ype of `plot`: `"p"` for observed points, `"l"` for fitted lines, `"b"` for both, and `"n"` for only setting axes.
`object`	Fitted result object.
`newdata`	Values of `mass` used in `predict`. The original data values are used if missing.
`fitted`	Fitted result object.
`parm`	Profiled parameters.
`alpha`, `maxsteps`, `del`	Parameters for profiling range and density (see Details).

Details

The no-interaction model assumes that the humped species richness pattern along biomass gradient is an artifact of plant size and density (Oksanen 1996). For low-biomass sites, it assumes that plants have a fixed size, and biomass increases with increasing number of plants. When the sites becomes crowded, the number of plants and species richness reaches the maximum. Higher biomass is reached by increasing the plant size, and then the number of plants and species richness will decrease. At biomasses below the hump, plant number and biomass are linearly related, and above the hump, plant number is proportional to inverse squared biomass. The number of plants is related to the number of species by the relationship (`link` afunction) from Fisher's log-series (Fisher et al. 1943).

The parameters of the model are:

1. `hump`: the location of the hump on the biomass gradient.
2. `scale`: an arbitrary multiplier to translate the biomass into virtual number of plants.
3. `alpha`: Fisher's α to translate the virtual number of plants into number of species.

The parameters `scale` and `alpha` are intermingled and this function should not be used for estimating Fisher's α. Probably the only meaningful and interesting parameter is the location of the `hump`.

Function may be very difficult to fit and easily gets trapped into local solutions, or fails with non-Poisson families, and function `profile` should be used to inspect the fitted models. You can use `plot.profile`, `pairs.profile` for graphical inspection of the profiles, and `confint` for the profile based confidence intervals. (With R prior to 4.4-0 you must use `library(MASS)` to access these functions.)

The original model intended to show that there is no need to speculate about “competition” and “stress” (Al-Mufti et al. 1977), but humped response can be produced as an artifact of using fixed plot size for varying plant sizes and densities.

Value

The function returns an object of class `"humpfit"` inheriting from class `"glm"`. The result object has specific `summary`, `predict`, `plot`, `points` and `lines` methods. In addition, it can be accessed by the following methods for `glm` objects: `AIC`, `extractAIC`, `deviance`, `coef`, `residuals.glm` (except `type = "partial"`), `fitted`, and perhaps some others. In addition, function `ellipse.glm` (package **ellipse**) can be used to draw approximate confidence ellipses for pairs of parameters, if the normal assumptions look appropriate.

Note

The function is a replacement for the original `GLIM4` function at the archive of Journal of Ecology. There the function was represented as a mixed `glm` with one non-linear parameter (`hump`) and a special one-parameter link function from Fisher's log-series. The current function directly applies non-linear maximum likelihood fitting using function `nlm`. Some expected problems with the current approach are:

- The function is discontinuous at `hump` and may be difficult to optimize in some cases (the lines will always join, but the derivative jumps).
- The function does not try very hard to find sensible starting values and can fail. The user may supply starting values in argument `start` if fitting fails.
- The estimation is unconstrained, but both `scale` and `alpha` should always be positive. Perhaps they should be fitted as logarithmic. Fitting `Gamma` `family` models might become easier, too.

References

Al-Mufti, M.M., Sykes, C.L, Furness, S.B., Grime, J.P & Band, S.R. (1977) A quantitative analysis of shoot phenology and dominance in herbaceous vegetation. *Journal of Ecology* 65,759–791.

Fisher, R.A., Corbet, A.S. & Williams, C.B. (1943) The relation between the number of species and the number of individuals in a random sample of of an animal population. *Journal of Animal Ecology* 12, 42–58.

Oksanen, J. (1996) Is the humped relationship between species richness and biomass an artefact due to plot size? *Journal of Ecology* 84, 293–295.

See Also

`fisherfit`, `profile.glm`, `confint.glm`.

Examples

```
mass <- c(140,230,310,310,400,510,610,670,860,900,1050,1160,1900,2480)
spno <- c(1,  4,  3,  9, 18, 30, 20, 14,  3,  2,  3,  2,  5,  2)
sol <- humpfit(mass, spno)
summary(sol) # Almost infinite alpha...
plot(sol)
## confint is in MASS, and impicitly calls profile.humpfit.
## Parameter 3 (alpha) is too extreme for profile and confint, and we
## must use only "hump" and "scale".
library(MASS)
plot(profile(sol, parm=1:2))
```

```
confint(sol, parm=c(1,2))
```

`iapq` *Number of Companion Species (IAP)*

Description

Function finds the number of companion species or the average species richness of other species for each species in a community. This is the quality index Q of Index of Atmospheric Purity (IAP) in lichen bioindication (LeBlanc & De Sloover 1970). The non-randomness of low or high Q values is found by randomization.

Usage

```
iapq(comm, freq.min = 5, permutations = 999)

iap(comm, iapq)

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

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

Arguments

`comm`	The community data frame.
`freq.min`	Minimum number of occurrences for analysed species.
`permutations`	Number of permutations to assess the randomized number of companion species.
`iapq`	Result of `iapq`.
`x`	`iapq` result object.
`...`	Other arguments to the function.
`object`	`iapq` result object.

Details

Index of Atmospheric Purity (IAP) is used in bioindication with epiphytic lichens and bryophytes (LeBlanc & De Sloover 1970). It derives species indicator scores (Q) as the number of other species in sampling units where each focal species is present, and then finds the IAP values for each sampling unit as scaled weighted sum of species indicator values.

Function `iapq` finds the Q values for all species in a community data set. Function `iap` applies these values for a community data set to evalute the IAP values for each site. The species are matched by names. LeBlanc & De Sloover (1970) used scaled abundance values and divided the weighted sum by 10, but this is not done in the current function, but this is left to the user.

Function `iapq` is a general measure of indicator value for species richness and can well be used outside lichen bioindication. The Q value is the average species richness in sampling units where the species is present, excluding the species itself from the richness. For rare species, Q is based on small sample size, and is therefore more variable than for common species. The `iapq` function assesses the non-randomness ('significance') of Q by taking random samples of the same size as the frequency (number of occurrence) of the focal species and finding the average richness (without the focal species) in these samples. Because species are more likely to be present in species-rich sampling units than in species-poor, the random sampling uses the observed species richness (with the focal species) as weights in random sampling. Testing is two-sided and the number of greater or less random values is multiplied with two. The observed value of Q is included in the random sample of species richness values both in assessing the p-value and in estimating the quantiles.

References

LeBlanc, S.C. & De Sloover, J. (1970) Relation between industrialization and the distribution and growth of epiphytic lichens and mosses in Montreal. *Can. J. Bot.* 48, 1485–1496.

`infoclust` *Information Clusterings of Ecological Communities*

Description

Function performs hierarchical clustering of binary ecological communities based on information analysis as defined by Williams et al. (1966) and Lance & Williams (1966).

Usage

```
infoclust(x, delta = TRUE)
```

Arguments

`x`	Community data
`delta`	Use increase in information (ΔI) instead of the value of information (I) when merging clusters as recommended by Williams et al. (1966) and Lance & Williams (1966).

Details

Function performs information analysis of binary ecological communities (Williams et al. 1966, Lance & Williams 1966). The current implementation is based on Legendre & Legendre (2012).

The information I of a collection of N sampling units with S species is defined as

$$I = SN \log N - \sum_i^S a_i \log a_i + (N - a_i) \log(N - a_i)$$

where a is the frequency count of each species in the collection. The method works by merging either the units that give the lowest increase (ΔI when `delta = TRUE`), or the units that are most homogeneous (lowest I when `delta = FALSE`). After merging sampling units or clusters, the community data matrix is updated by actually merging the data units and re-evaluating their information distance to all other units. The information content of all non-merged clusters is $I = 0$, and for clusters of several sampling units the constant species (completely absent or always present) do not contribute to the information. The largest increase in information is made by species with 0.5 relative frequency, so that the analysis tries to build clusters where species is either always present or always absent. This often gives easily interpretable clusters.

Value

Function returns an object of class `"infoclust"` that inherits from `hclust`. It uses all `"hclust"` methods, but some may fail or work in unexpected ways because the analysis is not based on dissimilarities but on binary data matrix.

References

Williams, W.T., Lambert, J.M. & Lance, G.N. (1966). Multivariate methods in plant ecology. V. Similarity analyses and information-analysis. *J. Ecol.* 54, 427–445.

Lance, G.N. & Williams, W.T. (1966). Computer programs for hierarchical polythetic classification ("similarity analyses"). *Comp. J.* 9, 60–64.

Legendre, P. & Legendre, L. (2012). *Numerical Ecology.* 3rd English Ed., Elsevier.

Examples

```
## example used to demonstrate the calculation of
## information analysis by Legendre & Legendre (2012, p. 372).
data(pond)
cl <- infoclust(pond)
plot(cl, hang = -1)
## Lance & Williams suggest a limit below which clustering is
## insignificant and should not be interpreted
abline(h=qchisq(0.95, ncol(pond)), col=2)
## Spurn Point Scrub data
data(spurn)
cl <- infoclust(spurn)
plot(cl, hang = -1)
if (require(vegan)) {
tabasco(spurn, cl)
## apply information clustering on species
tabasco(spurn, cl, infoclust(t(spurn)))
}
```

`mrankdist` *Mean Rank Shift Between Pairs of Sites*

Description

Function calculates the mean rank shift between decreasing rank orders of species (columns). The averaging is done by the number of species occurring in both compared sites, and the results are returned as dissimilarities.

Usage

```
mrankdist(x, missing = TRUE)
```

Arguments

`x`	The input data.
`missing`	If `TRUE`, missing species (zero abundance) will not be used in comparing rank shifts, but rank shifts are only compared for species that occur in both compared SUs. If `FALSE`, the missing species are given tied rank as last species.

References

Collins, S.L, Suding, K.N., Cleland, E.E., Batty, M., Pennings, S.C., Gross, K.L., Grace, J.B., Gough, L., Fargione, J.E. & Clark, C.M. (2008). Rank clocks and plant community dynamics. *Ecology* 89, 3534–3541.

`peelhull` *Convex Hull Enclosing a Given Proportion of Points*

Description

Functions find a small convex hull or ellipse enclosing a given proportion of points. The functions work by removing points from the hull or ellipse one by one. The points are selected either so that the area of the remaining hull is reduced as much as possible at every step (de Smith, Goodrich & Longley 2007), or removing the point that has the maximal total distance to all remaining points or longest Mahalanobis distance to the group.

Usage

```
peelhull(pts, keep = 0.9, criterion = c("area", "distance", "mahalanobis"))

peelellipse(pts, keep = 0.9, criterion = c("area", "mahalanobis"))
```

Arguments

`pts`	Coordinates of points, a two-column matrix
`keep`	Proportion of points kept
`criterion`	Criterion to remove a point on the hull. `"area"` removes the point that reduces the area most, `"distance"` remove the point at the hull that has the largest total distance to all points on and within the hull, and `"mahalanobis"` remove the point with longest Mahalanobis distance to the centroid.

Details

Reduction of area is the only criterion that really is based the area. The other methods are based on the distances to all points within and on the hull or on the Mahalanobis distance. The functions only work in 2D.

The algorithms are naïve, and stepwise removal of single points does not guarantee smallest possible final hull. Two outlier points close to each other can protect each other from removal, but removing both together would give a large reduction of the hull. The `"distance"` criterion produces circular hulls, but `"mahalanobis"` will better preserve the original elongation of the configuration, although it rarely gives smaller areas than `"area"` criterion.

Value

A two-column matrix of coordinates of peeled hull or ellipse defining that can be plotted with `polygon`, with attributes `"centre"` and `"area"`.

Author(s)

Jari Oksanen

References

de Smith, M.J., Goodchild, M.F. & Longley, P.A. (2007). *Geospatial analysis: A comprehensive guide to principles, techniques and software tools*. Matador.

Examples

```
x <- matrix(c(rnorm(100), rnorm(100, sd=2)), nrow=100, ncol=2)
op <- par(mar=c(0,0,1,0), mfrow=c(2,2))
## show first ten dropped points and hull keeping 50% of points
for (crit in c("area", "distance", "mahalanobis")) {
   plot(x, asp = 1, main = crit, xlab="", ylab="", axes=FALSE)
   for (p in c(100:90, 50)/100)
       polygon(peelhull(x, keep=p, crit=crit),
               col = adjustcolor("blue", alpha.f=0.04))
}
plot(x, asp = 1, main = "ellipse", xlab="", ylab="", axes=FALSE)
for (p in c(100:90, 50)/100)
   polygon(peelellipse(x, keep=p), col=adjustcolor("blue", alpha.f=0.04))

par(op) # original graphical parameters
```

`polarord` *Polar (Bray-Curtis) Ordination*

Description

Polar or Bray-Curtis ordination is a historic ordination method that could be performed without computers with simple hand calculations (Bray & Curtis 1957). Ordination axis is found by selecting two extreme points and projecting all points between these end points. The current function follows Beals (1984) in selecting the endpoints, projection of points on axis, and defining the residual distances for later axes.

Usage

```
polarord(d, k = 2, endpoints, metric = c("euclidean", "manhattan"))

## S3 method for class 'polarord'
plot(x, choices = c(1, 2), type = "t", display, ...)

## S3 replacement method for class 'polarord'
sppscores(object) <- value

## S3 method for class 'polarord'
eigenvals(x, ...)
```

Arguments

`d`	Dissimilarities or distances: a `dist` object.
`k`	Number of dimensions.
`endpoints`	Indices (not names) of endpoints for axes. This can be a single integer for the first endpoint, and the second endpoint and later axes are found automatically. If this is a vector, its non-zero values are taken as indices of endpoints of sequential axes.
`metric`	Use either `"euclidean"` or `"manhattan"` (city-block) metric in projecting points onto axes and calculating residual distances.
`x`	`polarord` result.
`choices`	Axes shown.
`type`	Type of graph which may be `"t"` for text, `"p"` for points or `"n"` for none (an empty plot).
`display`	Items displayed: `"sites"` are always available, but `"species"` only if they were added with sppscores.
`...`	Other arguments to the function (passed to `ordiplot`).
`object`	`polarord` result.
`value`	Community data to find the species scores.

Details

The implementation follows McCune & Grace (2002, Chapter 17). The endpoints are found using the variance-regression method of Beals (1984). The first endpoint has the highest variance of distances to other points. This guarantees that the point is at the margin of the multivariate cloud but is not an outlier, since outliers have long distances to all points and hence low variance of distances. The second endpoint has the lowest (most negative) regression coefficient between distances from the first and second point to all other points. This selects a point at the margin of the main cloud of points, opposite to the first endpoint. All points are projected on the axis between the endpoints, and this gives the scores on a polar ordination axis. Then the effect of the axis are removed by calculating residual distances. The projection of points and calculation of residuals is based either on Euclidean or Manhattan geometry, depending on the argument `metric`. Ecological indices are usually semimetric, and negative residual distances can emerge, but these are taken as zero in the current function.

The eigenvalues are estimated from the dispersion of points, and they are not necessarily in descending order. Usually only some first axes are reliable, and too high numbers of dimensions should be avoided. The inertia and eigenvalues give the average dispersion, and they are consistent with `dbrda` and `capscale`.

Polar ordination is a historical method that is little used today, but several authors claim that it is a powerful method (see McCune & Grace 2002). Although the basic operations can be easily performed by hand or graphically, the later developments of endpoint selection require more extensive calculations. With modern numerical utilities, the polar ordination is not faster than metric multidimensional scaling (`cmdscale`, `wcmdscale`).

It is possible to use predefined endpoints instead of automatic selection. This can be useful for confirmatory analysis (McCune & Grace 2002). If only one endpoint is given, the others are selected automatically.

McCune & Grace (2002) suggest that Polar Ordination can be alternatively performed with Manhattan or City-Block methods for projecting points onto axes and calculating residual ordination, although they say that this is "yet largerly untested". This can be implemented by setting `metric = "manhattan"`. However, two-dimensional configuration cannot be recovered from its Manhattan distances with this method, but it can be exactly recovered from its Euclidean distances and Euclidean metric (see Examples). Ordination is similar to the original two-dimensional configuration only if the endpoints were chosen so that the ordination axes are parallel to the dimensions in the original configuration. Further, ordinations with Manhattan metric are not invariant when rotated: Manhattan distances are calculated with respect to the axes, and they change if axes are rotated. This also means that fitted vectors cannot be used for environmental variables, and the interpretation of the ordination graph needs special and rare intuition. I warn against use of `metric = "manhattan"`.

Function `sppscores` can be used to add species scores to the ordination result.

Value

The function returns an object of class `"polarord"` with the following elements:

- `points`: The ordination scores.
- `inertia`: Total inertia of the input dissimilarities.
- `eig`: Eigenvalues of axes. These do not usually add up to total inertia and may not be in strictly descending order.
- `endpoints`: The indices (not the names) of the endpoints for each axis.
- `call`: Function call.

References

Beals, E. W. (1984) Bray-Curtis ordination: an effective strategy for analysis of ecological multivariate data. *Advances in Ecological Research* 14, 1–55.

Bray, J. R. & Curtis, J. T. (1957) An ordination of the upland forest communities in southern Wisconsin. *Ecological Monographs* 27, 325–349.

McCune, B. & Grace, J. B. (2002) *Analysis of Ecological Communities.* MjM Software Design.

Examples

```
data(spurn)
dis <- dist(spurn, method = "binary") ## Jaccard index
ord <- polarord(dis)
ord
summary(eigenvals(ord))
## add species scores
sppscores(ord) <- spurn
plot(ord)
## Two-dimensional configuration recovered with Euclidean metric
if (require(vegan)) { ## Procrustes analysis
set.seed(1009)
x <- matrix(runif(50), 25, 2) # 2-dim matrix
plot(procrustes(x, polarord(dist(x)))) # Euclidean: exact
plot(procrustes(x, polarord(dist(x, "man"), metric="man"))) # diverges
}
```

---

pond *Relative Zooplankton Abundances in Ponds*

Description

Relative abundances in scale 0 to 5 for un-named zooplankton species in ponds.

Usage

```
data("pond")
```

Format

A data frame with 5 ponds (observations) on 8 species (variables).

Details

This small data set was used to demonstrate dissimilarity and classification methods by Legendre & Legendre (2012).

Source

Legendre, P. & Legendre, L. (2012). *Numerical Ecology*. 3rd English Ed. Elsevier, page 290.

Examples

```
data(pond)
```

`posvectord` *Position Vector and Species Vector Ordination*

Description

Position Vector Ordination (PVO) finds ordination axes that go through sample points in species space and maximize the projections of other points onto them. These are similar to Principal Components. Species Vector Ordination (SVO) finds a set of species that maximize the projections of other species on species axes. This also can be similar to Principal Components and can be used to select a subset of species that cover most of the variance in the data.

Usage

```
posvectord(x, scale = FALSE)

spvectord(x, scale = FALSE)

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

Arguments

`x`	Input data.
`scale`	Scale variables to unit variance.
`...`	Other arguments (passed to `ordiplot`).

Details

Position Vector Ordination (PVO, function `posvectord`) is a simple educational method that resembles Principal Component Analysis, PCA (Orlóci 1966, 1973a). Function `posvectord` positions vectors from the data centroid (origin) to sample points in species space and evaluates the projections of other sample vectors on these positioned axes, and selects the one with highest total projection as the ordination axis. Then the effects of that selected axis are removed from the covariance matrix, zeroing the row and column of selected sampling unit, and the process is repeated. Principal Components maximize this projection, but PVO axes can be close to the Principal Component, in particular in the large data sets with many observations. The method was proposed as a computationally light approximation to PCA suitable for the computers of 1960s (Orlóci 1966). Now it can only be regarded as historically interesting, and also as an educational tool in introducing PCA.

Species Vector Ordination (SVO; function `spvectord`) is similar, but it picks the species vector that maximizes the species projected onto that vector (Orlóci 1973b). The method picks the species or variable that explains largest proportion of variance and uses the centred values of this variable as the axis. Then it orthogonalizes all species or variables to that selected axis and repeats the selection process. PCA finds a linear combination of species or variables that minimizes the residual variance, but in SVO only one species or variable is used. The axes are named by the species or variables, and the axis scores are the centred (residual) observed values. The resulting plot has orthogonalized set of species as axes. SVO can be similar to PCA, in particular when some few species contribute most of the the total variance. The method only has educational use in explaining PCA in species space. Orlóci (1973b) suggesed SVO as a method of selecting a subset of species or variables that contributes most of the variance in the data. The current `spvectord` function can also be used for this purpose, although it is mainly geared for ordination. **dave** package has function `orank` specifically written for variable or species selection using the same algorithm.

Value

`posvectord` returns an object of class `"posvectord"`, and `spvectord` returns an object of class `"spvectord"` that inherits from `"posvectord"`. Both result objects have the following elements:

- `points`: The ordination scores. In SVO, these are named by the species (variable) the axes is based on, the numerical scores are the centred (residual) values of observed data. In PVO, they are called as PVO, and they are scaled similarly as in PCA and can reconstitute (a low-rank approximation of) covariances.
- `totvar`: The total variance in the input data.
- `eig`: Eigenvalues of axes.

References

Orlóci, L. (1966) Geometric models in ecology. I. The theory and application of some ordination methods. *J. Ecol.* 54: 193–215.

Orlóci, L. (1973a) Ordination by resemblance matrices. In: R. H. Whittaker (ed.) *Ordination and Classification of Communities. Handbook of Vegetation Science* 5: 249–286.

Orlóci, L. (1973b) Ranking characters by dispersion criterion. *Nature* 244: 371–373.

See Also

`polarord` (Polar Ordination) is a similar educational tool to approximate Principal Coordinates Analysis (PCoA).

Examples

```
data(spurn)
spvectord(spurn)
m <- posvectord(spurn)
m
plot(m)
if (require(vegan, quietly = TRUE)) {
plot(procrustes(rda(spurn), m, choices=1:2))
}
```

qrao *Rao's Quadratic Entropy and Dissimilarity*

Description

Rao's quadratic entropy (Rao 1982) is a generalization of Simpson (or Gini-Simpson) diversity index to a situation where species are non-independent. The species dependences are expressed as dissimilarities, where 1 means independent species, and 0 a completely aliased species. Rao's distance (1982) is a similar generalization for distances between sampling units with non-independent species. Typically species distances are based on phylogenetics, taxonomy or species traits, and these measures are called phylogenetic or functional diversities and distances.

Function `raostand` modified data so that it is compatible to other functions, and Rao's quadratic entropy and Rao distances can be directly found from the standarized data.

Usage

```
qrao(x, d, na.rm = FALSE, dmax)

distrao(x, d, method = c("jensen", "euclidean", "standardized"), dmax)

raostand(x, d, propx = TRUE, dmax)
```

Arguments

`x`	Community data.
`d`	Phylogenetic distances or other dissimilarities among species.
`na.rm`	Should missing values be removed?
`dmax`	Scale dissimilarities by `dmax` (if `dmax` > `max(d)`) or truncate dissimilarities at `dmax` (if `dmax` < `max(d)`).
`method`	Distance measure to be used (see Details).
`propx`	Standardize row of `x` to unit total. This will give standardization that is compatible with `qrao` and `distrao`.

Details

Rao's quadratic entropy is $H_i = \sum_j \sum_k p_{ij} p_{ik} d_{jk}$, where i is the index of the sampling unit, j and k are indices of species, p is the proportion of species in the sampling unit, and d are distances among species. The distances should be scaled to range $0 \ldots 1$, and they are divided by the observed maximum if this exceeds 1. Alternatively, the distances are divided by argument `dmax` instead of data maximum. Distances that are shorter than `dmax` are truncated to the maximum value. The square roots of distances should be Euclidean, but this is not verified. They are Euclidean if there are no negative eigenvalues in the principal coordinates analysis, and **ade4** package has function `is.euclid` for a canned test. If all distances are 1, species are independent and `qrao` will return Simpson diversity.

Function `distrao` finds distances based on quadratic entropy for sampling units. Rao (1982) suggested distance $d_{ij} = H_{ij} - \frac{1}{2}(H_i + H_j)$, where H_i and H_j are Rao entropies for sites i and j and H_{ij} is the similar entropy evaluated so that the species proportions p are from different sampling units. Rao (1982) called this as Jensen distance, and it is half of the squared Euclidean distance. The Euclidean distance can also be requested. In addition, Rao (1982) suggested a standardized distance that is based on logarithms of elements H.

Function `raostand` standardizes data similarlty as implicitly done in `qrao` and `raodist` when `propx = TRUE`. For standardized data `Z`, quadratic entropy is found as `1 - rowSums(Z^2)`, and Rao distances can be found via Euclidean distances of `Z`. The standardized data allows calculating any generic community dissimilarity, and using `Z` in `rda` allows performing phylogenitically constrained RDA. The standardization does not preserve absences, but zero abundances will be boosted to positive values when the sampling unit has related species. More details can be found in vignette.

Value

`qrao` returns a vector of Rao's quadratic entropy values and `distrao` distances of class `"dist"`.

References

Rao, C.R. (1982) Diversity and dissimilarity coefficients: a unified approach. *Theoretical Population Biology* 21, 24–43.

See Also

There are other implementations of this function in R. Most notably functions `divc` and `disc` in **ade4**. However, these may square input dissimilarities and divide results by 2 depending on options. Function `taxondist` provides Clarke's taxonomic dissimilarity.

Examples

```
if (require(vegan)) {
data(dune, dune.phylodis)
qrao(dune, dune.phylodis)
tabasco(dune, hclust(distrao(dune, dune.phylodis)), hclust(dune.phylodis))
## regard lineages completely distinct beyond K/T (K/Pg) limit
qrao(dune, dune.phylodis, dmax = 65.2)
}

## Rao standardization
## Phylogenetic diversity
data(dune, dune.phylodis, dune.env)
Z <- raostand(dune, dune.phylodis)
all.equal(1 - rowSums(Z^2), qrao(dune, dune.phylodis),
     check.attributes = FALSE)
## Phylogenetic distance
all.equal(dist(Z), distrao(dune, dune.phylodis, method="euclidean"),
     check.attributes = FALSE)
## plot with standardized values
tabasco(Z, hclust(dist(Z)), hclust(dune.phylodis))
## phylogenetic polar ordination
pol <- polarord(vegdist(Z))
```

```
sppscores(pol) <- Z
plot(pol)
## Phylogenetically constrainted RDA
mod <- rda(raostand(dune, dune.phylodis, propx = FALSE) ~ Management + Moisture,
    dune.env)
anova(mod, by = "margin")
plot(mod, scaling = "sites")
```

`rdecorana` *Decorana: R implementation*

Description

Similar to Fortran implementation in vegan, but all in R.

Usage

```
rdecorana(
  x,
  iweigh = 0,
  iresc = 4,
  ira = 0,
  mk = 26,
  short = 0,
  before = NULL,
  after = NULL
)

## S3 method for class 'rdecorana'
scores(x, display = "sites", choices = 1:4, origin = FALSE, ...)
```

Arguments

`x`	`rdecorana` result object.
`iweigh`	Downweighting of rare species (0: no).
`iresc`	Number of rescaling cycles (0: no rescaling).
`ira`	Type of analyis (0: detrended, 1: orthogonal).
`mk`	Number of segments in detrending.
`short`	Shortest gradient to be rescaled.
`before, after`	Definition of Hill's piecewise transformation.
`display`	Scores for `"sites"` or `"species"`.
`choices`	Axes to returned.
`origin`	Return centred scores.
`...`	Other arguments passed to functions.

Details

This function duplicates **vegan** function decorana, but is written in R. It is slower, and does not have all features and support of tye **vegan** function, and there is no need to use this function for data analysis. The function serves two purposes. Firstly, as an R functin it is easier to inspect the algorithm than from C or Fortran code. Secondly, it is more hackable, and it is easier to develop new features, change code or replace functionality than in the compiled code. For instance, it would be trivial to add Detrended Constrained Correspondence Analysis, but this would be impossible without extensive changes in Fortran in the **vegan** function.

The main function rdecorana performs janitorial tasks, and for detrended CA it has basically only two loops. The first loop finds the next correspondence analysis axis for species and sites, and the second loop performs optional (if iresc > 0) nonlinear rescaling of that axis.

Two non-exported functions are used for finding the correspondence analysis axis: transvu runs one step of reciprocal averaging and the step is repeated so many times that the criterion value converges. Function transvu calls detrend that performs Hill's non-linear rescaling on mk segments. See Examples on alternative detrending methods. In all cases detrending against previous axis is done one axis by time starting from first, and then going again down to the first. For axis 4 the order of detrendings is 1, 2, 3, 2, 1. The criterion value is the one that **vegan** decorana calls 'Decorana values': for orthogonal CA is the eigenvalue, but for detrended CA it is a combination of eigenvalues and strength of detrending.

Non-linear rescaling is performed by function stretch that calls two functions: segment to estimate the dispersion of species, and smooth121 to smooth those estimates on segments (that number of segments is selected internally and mk has no influence. The criterion values for community data x with species and site scores v, u are based on $x_i j(u_i - v_j)^2$ summarized over sites i. Site scores u are weighted averages of species scores v, and therefore species scores should be symmetrically around corresponding site scores and the statistic describes their weighted dispersion. The purpose of rescaling is to make this dispersion 1 all over the axis. The procedure is complicated, and is best inspected from the code (which is commented).

Value

Currently returns a list of elements evals of Decorana values, with rproj and cproj of scaled row and column scores.

Row and column scores and the convergence criterion which for orthogonal CA is the eigenvalue.

Examples

```
data(spurn)
(mod <- rdecorana(spurn))
if (require(vegan)) {
## compare to decorana
decorana(spurn)
## ordiplot works
ordiplot(mod, display="sites")
}
## Examples on replacing the detrend() function with other
## alternatives. These need to use the same call as detrend() if we
## do not change the call in transvu.
##
```

```
## --- Smooth detrending
## detrend <- function(x, aidot, x1, mk)
##      residuals(loess(x ~ x1, weights=aidot))
## If you have this in the natto Namespace, you must use
## environment(detrend) <- environment(qrao) # any natto function
## assignInNamespace("detrend", detrend, "natto")
## ... but if you source() rdecorana.R, just do it!
## --- Quadratic polynomial detrending
## detrend <- function(x, aidot, x1, mk)
##       residuals(lm.wfit(poly(x1, 2), x, w = aidot))
## --- Orthogonal CA
## detrend <- function(x, aidot, x1, mk)
##       residuals(lm(x ~ x1, w = aidot))
## All these could handle all previous axes simultaneosly, but this
## requires editing transvu call.
```

recomm *Rebuild Communities from Rarefied Pieces*

Description

Community is divided into given number of pieces, and randomly rarefied community is derived for each of these pieces using `rrarefy`. The final community is a sum of rarefied pieces. The purpose is to mimick sampling variability of the observed community. However, the process loses rare species, and generated communities have lower species richness and diversity than the original one.

Usage

```
recomm(x, pieces = 4)
```

Arguments

`x`	communities to be rebuilt
`pieces`	number of pieces used to rebuild the community

Value

Randomized community data.

Author(s)

Jari Oksanen

`resassoc.cca` *Residual Species Associations in Constrained Ordination*

Description

Function finds residual species association after fitting the constraints (and conditions) in constrained ordination (`cca`, `rda`).

Usage

```
resassoc.cca(x, rank)
```

Arguments

`x`	Constrained ordination result from `cca` or `rda`.
`rank`	Number of unconstrained ordination axes that are used to compute the species associations. If missing, the rank is chosen by brokenstick distribution (`bstick`).

Details

The residual unconstrained axes can be used to assess if there is any important unexplained variation in constrained ordination. This approach was proposed already in the first papers on constrained ordination (ter Braak 1986). However, this has been rarely done, mainly because there are no easily available tools to assess the unconstrained axes. Bayesian analysis of species communities (Tikhonov et al. 2020) is conceptually similar to constrained ordination. There the random effects are analysed *via* correlated species responses that are interpreted to be caused by unknown environmental variables or residual inter-species association. The random effects are found as Bayesian latent factors which are conceptually similar to residual axes in constrained ordination (although these are maximum likelihood principal components). It is customary represent these residual correlations as ordered correlation matrix (Tikhonov et al 2017). This function provides similar tools for constrained correpsondence analysis (CCA) and redundancy analysis (RDA) as perfomed in **vegan** functions `cca` and `rda`.

In Bayesian framework, we only try to extract a low number of latent factors needed to describe correlated random variation. In constrained ordination, all residual variation is accounted for by residual axes. For meaningful analysis, we should only look at some first axes which may reflect systematic unexplained variation. With all axes, the residual correlations are usually nearly zero and systematic variation is masked by non-systematic noise. Therefore we should only have a look at some first axes. Users can either specify the number of axes used, and the default is to use brokenstick distribution (**vegan** function (`bstick`) to assess the number of axes that have surprisingly high eigenvalues and may reflect unaccounted systematic variation. The function returns inter-species associations as correlation-like numbers, where values nearly zero mean non-associated species. These values can be plotted in correlation plot or analysed directly. The correlations are directly found from the residual ordination axes.

It is an attractive idea to interpret inter-species association to describe ecological interactions among species (Tikhonov et el. 2017). This indeed is possible, but there are other alternative explanations, such as missing environmental variables or modelling errors (model formula, scaling of species, scaling of environmental variables, assumptions on the shape of species response etc.). You should be cautious in interpreting the results (and the same applies also to the analogous Bayesian model).

Value

Correlation-like association matrix between species with added argument `"rank"` of the unconstrained data used to compute associations.

References

ter Braak, C.J.F. (1986) Canonical correspondence analysis: a new eigenvector technique for multivariate direct gradient analysis. *Ecology* **67,** 1167–1179.

Tikhonov, G., Abrego, N., Dunson, D. and Ovaskainen, O. (2017) Using joint species distribution models for evaluating how species-to-species associations depend on the environmental context. *Methods in Ecology and Evolution* **8,** 443- 452.

Tikhonov et al. (2020) Joint species distribution modelling with the R-package Hmsc. *Methods in Ecology and Evolution* **11,** 442–447.

See Also

Hmsc has analogous function `computeAssociations`. However, in **Hmsc** the associations are based on Bayesian latent factors which are an essential natural component of the analysis whereas here we apply post-analysis tricks to extract something similar.

Examples

```
library(vegan)
data(dune, dune.env)
mod <- cca(dune ~ A1 + Moisture, dune.env)
resassoc.cca(mod)
```

`respthresh` *Binary Threshold for Continuous Fitted Response*

Description

Species distribution models (SDM) often use methods that give continuous fitted responses for the probability of species occurrence. Some users want to find a threshold that splits these continuous responses to binary outcomes of presence and absence (Allouche, Tsoar & Cadmon 2006). This function finds a threshold that maximizes the explained deviance and gives as true approximation of the original continuous response as possible.

Usage

```
respthresh(object)
```

Arguments

`object`	Fitted `glm` or `gam` object. The function was developed for binary observations responses fitted with `binomial` error distribution, but it may process other types of models with unspecified and untested results: proceed at your own risk.

Details

The function evaluates the deviance of the original continuous model by setting a threshold at each fitted value and evaluating the deviance explained with this threshold. The threshold is inclusive: values at or above the threshold are regarded as being hits. The deviance profile is usually jagged, and the method is sensitive to single data points, and can give disjunct regions of nearly equally good threshold points. Function `plot` will display the deviance profile, and `summary` lists all threshold that are nearly as good as the best point using Chi-square distribution with 1 degree of freedom at $p = 0.95$ as the criterion.

The binary model has two values or average responses below and at or above the threshold, but these values are not usually 0 and 1. However, they are the values that maximize the explained deviance of a two-value model.

The `summary` also gives a Deviance table showing the original Null deviance, the original model deviance, and between these the residual deviance with the binary threshold, and the differences of these deviances in the second model. The result object can also be accessed with `coef` that returns the threshold, `deviance` that returns the residual deviance, and `fitted` that returns the fitted two values for each original observation.

Value

The function returns an object of class `"respthresh"` with following elements:

- `coefficients`: optimal splitting threshold of fitted values.
- `expl.deviance`: explained deviance
- `deviance`: residual deviance at `threshold`.
- `cutoff, devprofile`: sorted possible thresholds (i.e., estimated fitted values) and associated explained deviances.
- `formula, null.deviance, orig.deviance`: `formula`, null deviance and deviance of the input model.
- `values`: fitted values below and above threshold.
- `fitted`: fitted values for each observation.

References

Allouche, O., Tsoar, A. & Kadmon, R. (2006) Assessing the accurraccy of species distribution models: prevelance, kappa and the true skill statistic (TSS). *Journal of Applied Ecology* 43, 1223–1232.

`sdca` *Smoothly Detrended Correspondence Analysis*

Description

Function `sdca` is similar to `decorana`, but instead of detrending by segments it uses `loess` for smooth non-linear detrending.

Usage

```
sdca(Y, iweigh = FALSE, pairwise = FALSE, monitor = FALSE, ...)
```

Arguments

`Y`	Input data.
`iweigh`	Downweight rare species.
`pairwise`	Detrend axis k separately for each previous axis in order $1 \ldots k-1 \ldots 1$. This only influences axes 3 and 4.
`monitor`	Turn on graphical monitoring of detrending for each axis.
`...`	Other arguments (passed to `loess`).

Details

Detrended correspondence analysis (DCA) as implemented in `decorana` tries to remove all systematic biases between ordination axes by detrending later axes against previous ones (Hill & Gauch 1980). DCA uses an ingenious method of detrending by axis segments to allow removing non-linear dependencies. Detrending means taking residuals against smoothed segment means as the new ordination scores for the current axis. It has been suggested that abrupt changes at segment borders can cause some problems in DCA. The current function replaces segmented detrending with detrending against smooth `loess` functions. However, in many cases this changes the results little from the original detrending by segments.

The detrending for axes 3 and 4 is perfomed either using all previous axes simultaneously in `loess` (default) or if `pairwise=TRUE` performing sequential separate detrending against each previous axis separately so that both the first and last detrending are performed against axis 1 similarly as in the original `decorana`. For axis 3 the sequence is against axes 1, 2, 1 and for axis 4 against axes 1, 2, 3, 2, 1.

The `decorana` software made several other innovations than detrending. Rescaling of axes is often more influential in application than the actual detrending, like you can see by using `decorana` without rescaling.

Value

Function returns a subset of `decorana` result object and can use many `decorana` methods (such as `plot`).

References

Hill, M.O. and Gauch, H.G. (1980). Detrended correspondence analysis: an improved ordination technique. *Vegetatio* **42**, 47–58.

Examples

```
data(spurn)
mod <- sdca(spurn)
plot(mod, display="species")
if (require(vegan)) {
## compare against original decorana without rescaling
mod0 <- decorana(spurn, iresc = 0)
plot(procrustes(mod0, mod, choices=1:2))
}
```

---

slopediff *Compare Slopes of Mantel Regression Within and Across Blocks*

Description

Functions fit linear regression on distances within blocks and across blocks and compere their slopes using permutation tests.

Usage

```
slopediff(ydist, xdist, block, nperm = 999)

slopediff2(ydist, xdist, block, nperm = 999)

## S3 method for class 'slopediff'
plot(x, col = 1, obsline = TRUE, ...)

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

Arguments

`ydist, xdist`	Dependent and indpendent dissimilarities.
`block`	Blocking of sites.
`nperm`	Number of simple permutations.
`x`	Result output from `slopediff` or `slopediff2`.
`col`	Plotting colour.
`obsline`	Draw line on observed statistic in the `density` plot.
`...`	Other parameters passed to `plot`.
`object`	Result object from `slopediff` or `slopediff2`.

Details

Functions are intended to analyse the differences of linear Mantel regression within and across blocks. Typically the dependent dissimilarities (`ydist`) are based on community data, independent dissimilarities (`xdist`) are based on environmental data, and `block` defines subsets of data, such as geographical subareas.

Functions `slopediff` and `slopediff2` differ in their permutation models. Function `slopediff` permutes the classification vector defining `blocks`, and keeps the dependent (`ydist`) and independent (`xdist`) pairs of dissimilarities fixed. Function `slopediff2` permutes dependent data, and keeps `blocks` and dependent distances fixed.

Author(s)

Jari Oksanen

`spurn` *Dune Scrub Vegetation in Spurn Point, Yorkshire*

Description

Shimwell (1971) used the Spurn Point Dune Scrub data as an example well suited for Braun-Blanquet tabular rearrangement method.

Usage

```
data("spurn")
```

Format

A data frame with 20 sample plots on the following 40 species.

`Elaerham`	*Elaeagnus rhamnoides* (as *Hippophae*)
`Jacovulg`	*Jacobaea vulgaris* (as *Senecio jacobaea*)
`Soladulc`	*Solanum dulcamara*
`Rubufrut`	*Rubus fruticosus* s.lat.
`UrtidioiA`	*Urtica dioica*
`Rumecris`	*Rumex crispus*
`ClayperfB`	*Claytonia perfoliata* (as *Montia*)
`StelmediB`	*Stellaria media*
`FestrubrC`	*Festuca rubra*
`ElymrepeC`	*Elymus repens* (as *Agropyron*)
`AmmoarenC`	*Ammophila arenaria*
`Soncarve`	*Sonchus arvensis*
`Ononspin`	*Ononis spinosa* (as *O. repens*)

`Galiveru`	*Galium verum*
`Calysold`	*Calystegia soldanella*
`PoapratC`	*Poa pratensis*
`Agrostol`	*Agrostis stolonifera*
`RanubulbC`	*Ranunculus bulbosus*
`PlanlancC`	*Plantago lanceolata*
`Verocham`	*Veronica chamaedrys*
`Epilangu`	*Epilobium angustifolium* (as *Chamaenerion*)
`CerafontB`	*Cerastium fontanum* (as *C. vulgatum*)
`Sambnigr`	*Sambucus nigra*
`CirsvulgB`	*Cirsium vulgare*
`Heraspho`	*Heracleum sphondylium*
`Inulcony`	*Inula conyza*
`CardhirsB`	*Cardamine hirsuta*
`Hyporadi`	*Hypochaeris radicata*
`Arrhelat`	*Arrhenatherum elatius*
`Soncaspe`	*Sonchus asper*
`EurhpraeA`	*Eurhynchium praelongum*
`Hypncupr`	*Hypnum cupressiforme*
`Bracruta`	*Brachythecium rutabulum*
`GeasfornB`	*Geastrum fornicatum*
`BracalbiC`	*Brachythecium albicans*
`Bryuincl`	*Bryum inclinatum*
`Syntrura`	*Syntrichia ruralis* (as *Tortula ruraliformis*)
`Cladranf`	*Cladonia rangiformis*
`Bovinigr`	*Bovista nigrescens*
`Lophhete`	*Lophocolea heterophylla*

The species abundances were visually assessed with Braun-Blanquet scale values +, 1 . . . 5 which are presented in numeric scale 1 . . . 6 in the data set.

Shimwell (1971) used the data set to demonstrate the Braun-Blanquet tabular rearrangement method. He divided the data into three classification units marked as A, B and C, and removed one data set marked as X. The sample plots are named with consecutive numbers after a letter corresponding to these classes. The diagnostic taxa have a corresponding upper case letter after their name.

Source

Shimwell, D. W. (1971) *Description and Classification of Vegetation (Sidgwick & Jackson), Table 44.*

Examples

```
data(spurn)
```

`subdiag` *Extract Subdiagonal from Dissimilarities or from a Square Matrix*

Description

Extract Subdiagonal from Dissimilarities or from a Square Matrix

Usage

```
subdiag(x)
```

Arguments

`x`	Square matrix or a `dist` object.

`swanrandom` *Generate Communities as Random Poisson from Swan Expectations*

Description

Poisson random numbers lose species as they introduce new zeros for low expected values. They are also unable to compensate this by generating new "unseen" species. To compensate this, we first use Swan transformation with one pass to generate above-zero expected values for unseen species. After generating the Swan probabilities for missing species we standardize all species to their original totals; this is a similar idea as the species coverage in the Good-Turing model (Good 1953). Finally, a community is generated as a Poisson random variate of adjusted observed abundances and Swan probabilities. The process maintains average species richness and diversity, but Poisson distribution probably underestimates the real sampling variance.

Usage

```
swanrandom(x)
```

Arguments

`x`	community data of integer counts

Value

Randomized community matrix.

Author(s)

Jari Oksanen

References

Good, I.J. (1953) The population frequencies of species and the estimation of population parameters. Biometrika 40, 237–264.

`taxondist` *Clarke's Taxonomic Community Dissimilarity*

Description

Taxonomic dissimilarity takes into account related species when there is no exact species match in compared communities. This function returns generalizations of Bray-Curtis and Kulczynski indices (Clarke _et al._ 2006).

Usage

```
taxondist(x, d, method = c("gamma", "theta"), dmax)
```

Arguments

`x`	Community data; will be treated as binary presence/absence matrix.
`d`	Taxonomic, phylogenetic or other dissimilarities among species (columns of `x`).
`method`	Type of returned dissimilarity index. Gamma is generalized Bray-Curtis, and Theta generalized Kulczynski index.
`dmax`	Scale dissimilarities by `dmax` (if `dmax` > `max(d)`) or truncate dissimilarities at `dmax` (if `dmax` < `max(d)`).

Details

When a species occurs only in one of two compared communities, the species will increase dissimilarity of two communities with one species. However, if the other communities contains related species, the increase will be less depending on the relatedness of species (Clarke _et al._ 2006). The degree of relatedness is defined by taxonomic or other taxon dissimilarities.

The current function generalizes Bray-Curtis and Kulczynski indices (see `canneddist`) for binary (presence-absence) data by taking the dissimilarity to most closely related species as the increase of dissimilarity, instead of 1 of basic index. If species dissimilarities have any values over 1, they will be scaled by observed maximum. Alternatively, user can specify maximum species dissimilarity (argument ‘dmax‘) for scaling, and if it is lower than data maximum, higher values will be truncated to scaled value 1. This allows imposing stricter concept of relatedness so that, say, taxa in different orders will be regarded as completely unrelated.

Although the function was proposed and is named as taxonomic dissimilarity, the function can be used with phylogenetic, functional or trait dissimilarities.

Function `distrao` provides an alternative index that can handle quantitative data and produce indices related to Euclidean distance.

Value

Clarke's taxonomic dissimilarity index as defined in `method`.

References

Clarke, K.R., Somerfield, P.J. & Chapman, M.G. (2006). On resemblance measures for ecological studies, including taxonomic dissimilarities and a zero-adjusted Bray-Curtis coefficient for denuded assemblages. _J. Exp. Marine Biol. & Ecol._ 330, 55-80.

See Also

`distrao` is an alternative taxonomic distance measure. The index is closely related to Clarke's taxonomic diversity indices which are available in **vegan** function `taxondive`. **Vegan** function `taxa2dist` can be used to find taxonomic dissimilarities from classification table.

Examples

```
if (require(vegan)) {
data(dune, dune.phylodis)
## phylogenetic data, but regard lineaages completely distict
## beyond K/T (K/Pg) age limit
d <- taxondist(dune, dune.phylodis, dmax = 65.2)
polarord(d)
}
```

Index