The vegan Package

February 9, 2009

Title Community Ecology Package

Version 1.15-1

Date December 19, 2008

Author Jari Oksanen, Roeland Kindt, Pierre Legendre, Bob O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, Helene Wagner

Maintainer Jari Oksanen <jari.oksanen@oulu.fi>

Suggests MASS, mgcv, lattice, cluster, scatterplot3d, rgl, ellipse, tcltk

Description Ordination methods, diversity analysis and other functions for community and vegetation ecologists.

License GPL-2

URL http://cran.r-project.org/, http://vegan.r-forge.r-project.org/

R topics documented:

`BCI` *Barro Colorado Island Tree Counts*

Description

Tree counts in 1-hectare plots in the Barro Colorado Island.

Usage

```
data(BCI)
```

Format

A data frame with 50 plots (rows) of 1 hectare with counts of trees on each plot with total of 225 species (columns). Full Latin names are used for tree species.

Details

Data give the numbers of trees at least 10 cm in diameter at breast height (1.3 m above the ground) in each one hectare square of forest. Within each one hectare square, all individuals of all species were tallied and are recorded in this table.

The data frame contains only the Barro Colorado Island subset of the original data.

The quadrats are located in a regular grid. See`examples` for the coordinates.

Source

http://www.sciencemag.org/cgi/content/full/295/5555/666/DC1

References

Condit, R, Pitman, N, Leigh, E.G., Chave, J., Terborgh, J., Foster, R.B., Nuñez, P., Aguilar, S., Valencia, R., Villa, G., Muller-Landau, H.C., Losos, E. & Hubbell, S.P. (2002). Beta-diversity in tropical forest trees. *Science* 295, 666–669.

See Also

`BCI.env` in **BiodiversityR** package for environmental data (coordinates are given below in the `examples`).

Examples

```
data(BCI)
## UTM Coordinates (in metres)
UTM.EW <- rep(seq(625754, 626654, by=100), each=5)
UTM.NS <- rep(seq(1011569,  1011969, by=100), len=50)
```

`CCorA` *Canonical Correlation Analysis*

Description

Canonical correlation analysis, following Brian McArdle's unpublished graduate course notes, plus improvements to allow the calculations in the case of very sparse and collinear matrices.

Usage

```
CCorA(Y, X, stand.Y=FALSE, stand.X=FALSE, nperm = 0, ...)

## S3 method for class 'CCorA':
biplot(x, xlabs, which = 1:2, ...)
```

Arguments

`Y`	left matrix.
`X`	right matrix.
`stand.Y`	logical; should `Y` be standardized?
`stand.X`	logical; should `X` be standardized?
`nperm`	numeric; Number of permutations to evaluate the significance of Pillai's trace
`x`	`CCoaR` result object
`xlabs`	Row labels. The default is to use row names, `NULL` uses row numbers instead, and `NA` suppresses plotting row names completely
`which`	`1` plots `Y` reseults, and `2` plots `X1` results
`...`	Other arguments passed to functions. `biplot.CCorA` passes graphical arguments to `biplot` and `biplot.default`, `CCorA` currently ignores extra arguments.

Details

Canonical correlation analysis (Hotelling 1936) seeks linear combinations of the variables of `Y` that are maximally correlated to linear combinations of the variables of `X`. The analysis estimates the relationships and displays them in graphs.

Algorithmic notes:

1. All data matrices are replaced by their PCA object scores, computed by SVD.
2. The blunt approach would be to read the three matrices, compute the covariance matrices, then the matrix `S12 %*% inv(S22) %*% t(S12) %*% inv(S11)`. Its trace is Pillai's trace statistic.
3. This approach may fail, however, when there is heavy multicollinearity in very sparse data matrices, as it is the case in 4th-corner inflated data matrices for example. The safe approach is to replace all data matrices by their PCA object scores.
4. Inversion by `solve` is avoided. Computation of inverses is done by SVD (`svd`) in most cases.
5. Regression by OLS is also avoided. Regression residuals are computed by QR decomposition (`qr`).

The `biplot` function can produce two biplots, each for the left matrix and right matrix solutions. The function passes all arguments to `biplot.default`, and you should consult its help page for configuring biplots.

Value

Function `CCorA` returns a list containing the following components:

`Pillai`	Pillai's trace statistic = sum of canonical eigenvalues.
`EigenValues`	Canonical eigenvalues. They are the squares of the canonical correlations.
`CanCorr`	Canonical correlations.
`Mat.ranks`	Ranks of matrices `Y` and X1 (possibly after controlling for X2).
`RDA.Rsquares`	Bimultivariate redundancy coefficients (R-squares) of RDAs of Y|X1 and X1|Y.
`RDA.adj.Rsq`	`RDA.Rsquares` adjusted for n and number of explanatory variables.
`AA`	Scores of Y variables in Y biplot.
`BB`	Scores of X1 variables in X1 biplot.
`Cy`	Object scores in Y biplot.
`Cx`	Object scores in X1 biplot.

Author(s)

Pierre Legendre, Departement de Sciences Biologiques, Universite de Montreal. Implemented in **vegan** with the help of Jari Oksanen.

References

Hotelling, H. 1936. Relations between two sets of variates. *Biometrika* **28**: 321-377.

Examples

```
# Example using random numbers
mat1 <- matrix(rnorm(60),20,3)
mat2 <- matrix(rnorm(100),20,5)
CCorA(mat1, mat2)

# Example using intercountry life-cycle savings data, 50 countries
data(LifeCycleSavings)
pop <- LifeCycleSavings[, 2:3]
oec <- LifeCycleSavings[, -(2:3)]
out <- CCorA(pop, oec)
out
biplot(out, xlabs = NA)
```

---

add1.cca — *Add or Drop Single Terms to a Constrained Ordination Model*

Description

Compute all single terms that can be added or dropped from a constrained ordination model.

Usage

```
## S3 method for class 'cca':
add1(object, scope, test = c("none", "permutation"),
    pstep = 100, perm.max = 200, ...)
## S3 method for class 'cca':
drop1(object, scope, test = c("none", "permutation"),
    pstep = 100, perm.max = 200, ...)
```

Arguments

`object`	A constrained ordination object from `cca`, `rda` or `capscale`.
`scope`	A formula giving the terms to be considered for adding or dropping; see `add1` for details.
`test`	Should a permutation test added using `anova.cca`.
`pstep`	Number of permutations in one step, passed as argument `step` to `anova.cca`.
`perm.max`	Maximum number of permutation in `anova.cca`.
`...`	Other arguments passed to `add1.default`, `drop1.default`, and `anova.cca`.

Details

With argument `test = "none"` the functions will only call `add1.default` or `drop1.default`. With argument `test = "permutation"` the functions will add test results from `anova.cca`. Function `drop1.cca` will call `anova.cca` with argument `by = "margin"`. Function `add1.cca` will implement a test for single term additions that is not directly available in `anova.cca`.

Functions are used implicity in `step`. The `deviance.cca` and `deviance.rda` used in `step` have no firm basis, and setting argument `test = "permutation"` may help in getting useful insight into validity of model building. Meticulous use of `add1.cca` and `drop1.cca` will allow more judicious model building.

The default `perm.max` is set to a low value, because permutation tests can take a long time. It should be sufficient to give a impression on the significances of the terms, but higher values of `perm.max` should be used if P values really are important.

Value

Returns a similar object as `add1` and `drop1`.

Author(s)

Jari Oksanen

See Also

`add1`, `drop1` and `anova.cca` for basic methods. You probably need these functions with `step`. Functions `deviance.cca` and `extractAIC.cca` are used to produce the other arguments than test results in the output. Functions `cca`, `rda` and `capscale` produce result objects for these functions.

Examples

```
data(varespec)
data(varechem)
step(rda(varespec ~  1, varechem), reformulate(names(varechem)), test="perm")
```

adonis — *Permutational Multivariate Analysis of Variance Using Distance Matrices*

Description

Analysis of variance using distance matrices — for partitioning distance matrices among sources of variation and fitting linear models (e.g., factors, polynomial regression) to distance matrices; uses a permutation test with pseudo-F ratios.

Usage

```
adonis(formula, data, permutations = 5, method = "bray",
       strata = NULL, contr.unordered = "contr.sum",
       contr.ordered = "contr.poly", ...)
```

Arguments

`formula`	a typical model formula such as `Y ~ A + B*C`, but where `Y` is either a dissimilarity object (inheriting from class `"dist"`) or data frame or a matrix; `A`, `B`, and `C` may be factors or continuous variables. If a dissimilarity object is supplied, no species cofficients can be calculated (see Value below).
`data`	the data frame from which `A`, `B`, and `C` would be drawn.
`permutations`	number of replicate permutations used for the hypothesis tests (F tests).
`method`	the name of any method used in `vegdist` to calculate pairwise distances if the left hand side of the `formula` was a data frame or a matrix.
`strata`	groups (strata) within which to constrain permutations.
`contr.unordered, contr.ordered`	contrasts used for the design matrix (default in R is dummy or treatment contrasts for unordered factors).
`...`	Other arguments passed to `vegdist`.

Details

`adonis` is a function for the analysis and partitioning sums of squares using semimetric and metric distance matrices. Insofar as it partitions sums of squares of a multivariate data set, it is directly analogous to MANOVA (multivariate analysis of variance). M.J. Anderson (McArdle and Anderson 2001, Anderson 2001) refers to the method as “permutational manova” (formerly “nonparametric manova”). Further, as its inputs are linear predictors, and a response matrix of an arbitrary number of columns (2 to millions), it is a robust alternative to both parametric MANOVA and to ordination methods for describing how variation is attributed to different experimental treatments or uncontrolled covariates. It is also analogous to redundancy analysis (Legendre and Anderson 1999).

Typical uses of `adonis` include analysis of ecological community data (samples X species matrices) or genetic data where we might have a limited number of samples of individuals and thousands or millions of columns of gene expression data (e.g. Zapala and Schork 2006).

`adonis` is an alternative to AMOVA (nested analysis of molecular variance, Excoffier, Smouse, and Quattro, 1992; `amova` in the **ade4** package) for both crossed and nested factors.

Like AMOVA (Excoffier et al. 1992), `adonis` relies on a long-understood phenomenon that allows one to partition sums of squared deviations from a centroid in two different ways (McArdle and Anderson 2001). The most widely recognized method, used, e.g., for ANOVA and MANOVA, is to first identify the relevant centroids and then to calculated the squared deviations from these points. For a centered $n \times p$ response matrix Y, this method uses the $p \times p$ inner product matrix $Y'Y$. The less appreciated method is to use the $n \times n$ outer product matrix YY'. Both AMOVA and `adonis` use this latter method. This allows the use of any semimetric (e.g. Bray-Curtis, aka Steinhaus, Czekanowski, and Sørensen) or metric (e.g. Euclidean) distance matrix (McArdle and Anderson 2001). Using Euclidean distances with the second method results in the same analysis as the first method.

Significance tests are done using F-tests based on sequential sums of squares from permutations of the raw data, and not permutations of residuals. Permutations of the raw data may have better small sample characteristics. Further, the precise meaning of hypothesis tests will depend upon precisely what is permuted. The strata argument keeps groups intact for a particular hypothesis test where one does not want to permute the data among particular groups. For instance, `strata = B` causes permutations among levels of `A` but retains data within levels of `B` (no permutation among levels of `B`).

The default `contrasts` are different than in R in general. Specifically, they use "sum" contrasts, sometimes known as "ANOVA" contrasts. See a useful text (e.g. Crawley, 2002) for a transparent introduction to linear model contrasts. This choice of contrasts is simply a personal pedagogical preference. The particular contrasts can be set to any `contrasts` specified in R, including Helmert and treatment contrasts.

Rules associated with formulae apply. See "An Introduction to R" for an overview of rules.

`print.adonis` shows the `aov.tab` component of the output.

Value

This function returns typical, but limited, output for analysis of variance (general linear models).

`aov.tab`	Typical AOV table showing sources of variation, degrees of freedom, sequential sums of squares, mean squares, F statistics, partial R-squared and P values, based on N permutations.
`coefficients`	matrix of coefficients of the linear model, with rows representing sources of variation and columns representing species; each column represents a fit of a species abundance to the linear model. These are what you get when you fit one species to your predictors. These are NOT available if you supply the distance matrix in the formula, rather than the site x species matrix
`coef.sites`	matrix of coefficients of the linear model, with rows representing sources of variation and columns representing sites; each column represents a fit of a sites distances (from all other sites) to the linear model.These are what you get when you fit distances of one site to your predictors.
`f.perms`	an N by m matrix of the null F statistics for each source of variation based on N permutations of the data.

Author(s)

Martin Henry H. Stevens ⟨HStevens@muohio.edu⟩, adapted to **vegan** by Jari Oksanen.

References

Anderson, M.J. 2001. A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, **26**: 32–46.

Crawley, M.J. 2002. *Statistical Computing: An Introduction to Data Analysis Using S-PLUS*

Excoffier, L., P.E. Smouse, and J.M. Quattro. 1992. Analysis of molecular variance inferred from metric distances among DNA haplotypes: Application to human mitochondrial DNA restriction data. *Genetics*, **131**:479–491.

Legendre, P. and M.J. Anderson. 1999. Distance-based redundancy analysis: Testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs*, **69**:1–24.

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

Zapala, M.A. and N.J. Schork. 2006. Multivariate regression analysis of distance matrices for testing associations between gene expression patterns and related variables. *Proceedings of the National Academy of Sciences, USA*, **103**:19430–19435.

See Also

`mrpp`, `anosim`, `mantel`, `varpart`.

Examples

```
data(dune)
data(dune.env)
adonis(dune ~ Management*A1, data=dune.env, permutations=100)
```

`anosim` *Analysis of Similarities*

Description

Analysis of similarities (ANOSIM) provides a way to test statistically whether there is a significant difference between two or more groups of sampling units.

Usage

```
anosim(dis, grouping, permutations=1000, strata)
```

Arguments

`dis`	Dissimilarity matrix.
`grouping`	Factor for grouping observations.
`permutations`	Number of permutation to assess the significance of the ANOSIM statistic.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.

Details

Analysis of similarities (ANOSIM) provides a way to test statistically whether there is a significant difference between two or more groups of sampling units. Function `anosim` operates directly on a dissimilarity matrix. A suitable dissimilarity matrix is produced by functions `dist` or `vegdist`. The method is philosophically allied with NMDS ordination (`isoMDS`), in that it uses only the rank order of dissimilarity values.

If two groups of sampling units are really different in their species composition, then compositional dissimilarities between the groups ought to be greater than those within the groups. The `anosim` statistic R is based on the difference of mean ranks between groups (r_B) and within groups (r_W):

$$R = (r_B - r_W)/(N(N-1)/4)$$

The divisor is chosen so that R will be in the interval $-1 \ldots +1$, value 0 indicating completely random grouping.

The statistical significance of observed R is assessed by permuting the grouping vector to obtain the empirical distribution of R under null-model.

The function has `summary` and `plot` methods. These both show valuable information to assess the validity of the method: The function assumes that all ranked dissimilarities within groups have about equal median and range. The `plot` method uses `boxplot` with options `notch=TRUE` and `varwidth=TRUE`.

Value

The function returs a list of class `"anosim"` with following items:

`call`	Function call.
`statistic`	The value of ANOSIM statistic R
`signif`	Significance from permutation.
`perm`	Permutation values of R
`class.vec`	Factor with value `Between` for dissimilarities between classes and class name for corresponding dissimilarity within class.
`dis.rank`	Rank of dissimilarity entry.
`dissimilarity`	The name of the dissimilarity index: the `"method"` entry of the `dist` object.

Note

I don't quite trust this method. Somebody should study its performance carefully. The function returns a lot of information to ease further scrutiny. Most `anosim` models could be analysed with `adonis` which seems to be a more robust alternative.

Author(s)

Jari Oksanen, with a help from Peter R. Minchin.

References

Clarke, K. R. (1993). Non-parametric multivariate analysis of changes in community structure. *Australian Journal of Ecology* 18, 117-143.

See Also

mrpp for a similar function using original dissimilarities instead of their ranks. dist and vegdist for obtaining dissimilarities, and rank for ranking real values. For comparing dissimilarities against continuous variables, see mantel. Function adonis is a more robust alternative that should preferred.

Examples

```
data(dune)
data(dune.env)
dune.dist <- vegdist(dune)
attach(dune.env)
dune.ano <- anosim(dune.dist, Management)
summary(dune.ano)
plot(dune.ano)
```

---

anova.cca *Permutation Test for Constrained Correspondence Analysis, Redundancy Analysis and Constrained Analysis of Principal Coordinates*

Description

The function performs an ANOVA like permutation test for Constrained Correspondence Analysis (cca), Redundancy Analysis (rda) or Constrained Analysis of Principal Coordinates (capscale) to assess the significance of constraints.

Usage

```
## S3 method for class 'cca':
anova(object, alpha=0.05, beta=0.01, step=100, perm.max=9999,
      by = NULL, ...)

permutest(x, ...)

## S3 method for class 'cca':
permutest(x, permutations = 100,
          model = c("reduced", "direct", "full"),
          first = FALSE, strata, ...)
```

Arguments

object,x	A result object from cca.
alpha	Targeted Type I error rate.
beta	Accepted Type II error rate.
step	Number of permutations during one step.
perm.max	Maximum number of permutations.

`by`	Setting `by = "axis"` will assess significance for each constrained axis, and setting `by = "terms"` will assess significance for each term (sequentially from first to last), and setting `by = "margin"` will assess the marginal effects of the terms (each marginal term analysed in a model with all other variables).
`...`	Parameters to `permutest.cca`.
`permutations`	Number of permutations for assessing significance of constraints.
`model`	Permutation model (partial match).
`first`	Assess only the significance of the first constrained eigenvalue; will be passed from `anova.cca`.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.

Details

Functions `anova.cca` and `permutest.cca` implement an ANOVA like permutation test for the joint effect of constraints in `cca`, `rda` or `capscale`. Functions `anova.cca` and `permutest.cca` differ in printout style and in interface. Function `permutest.cca` is the proper workhorse, but `anova.cca` passes all parameters to `permutest.cca`.

In `anova.cca` the number of permutations is controlled by targeted "critical" P value (`alpha`) and accepted Type II or rejection error (`beta`). If the results of permutations differ from the targeted `alpha` at risk level given by `beta`, the permutations are terminated. If the current estimate of P does not differ significantly from `alpha` of the alternative hypothesis, the permutations are continued with `step` new permutations (at the first step, the number of permuations is `step - 1`). However, with `by = "terms"` a fixed number of permutations will be used, and this is given by argument `permutations`, or if this is missing, by `step`.

The function `permutest.cca` implements a permutation test for the "significance" of constraints in `cca`, `rda` or `capscale`. Community data are permuted with choice `model = "direct"`, residuals after partial CCA/RDA/CAP with choice `model = "reduced"` (default), and residuals after CCA/RDA/CAP under choice `model = "full"`. If there is no partial CCA/RDA/CAP stage, `model = "reduced"` simply permutes the data and is equivalent to `model = "direct"`. The test statistic is "pseudo-F", which is the ratio of constrained and unconstrained total Inertia (Chi-squares, variances or something similar), each divided by their respective ranks. If there are no conditions ("partial" terms), the sum of all eigenvalues remains constant, so that pseudo-F and eigenvalues would give equal results. In partial CCA/RDA/CAP, the effect of conditioning variables ("covariables" is removed before permutation, and these residuals are added to the non-permuted fitted values of partial CCA (fitted values of `X ~ Z`). Consequently, the total Chi-square is not fixed, and test based on pseudo-F would differ from the test based on plain eigenvalues. CCA is a weighted method, and environmental data are re-weighted at each permutation step using permuted weights.

The default test is for the sum of all constrained eigenvalues. Setting `first = TRUE` will perform a test for the first constrained eigenvalue. Argument `first` can be set either in `anova.cca` or in `permutest.cca`. It is also possible to perform significance tests for each axis or for each term (constraining variable) using argument `by` in `anova.cca`. Setting `by = "axis"` will perform separate significance tests for each constrained axis. All previous constrained axes will be used as conditions ("partialled out") and a test for the first constrained eigenvalues is performed. Setting `by = "terms"` will perform separate significance test for each term (constraining variable). The terms are assessed sequentially from first to last, and the order of the terms will influence their significances. Setting `by = "margin"` will perform separate significance test for each marginal term in a model with all other terms. The marginal test also accepts a `scope` argument for the `drop.scope` which can be a character vector of term labels that are analysed, or a fitted model of lower scope. The marginal effects are also known as "Type III" effects, but the current function only evaluates marginal terms. It will, for instance, ignore main effects that are included in interaction terms. In calculating pseudo-F, all terms are compared to the same residual of the full model. Permutations for all axes or terms will start from the same `.Random.seed`, and the seed will be advanced to the value after the longest permutation at the exit from the function.

Value

Function `permutest.cca` returns an object of class `"permutest.cca"`, which has its own `print` method. The function `anova.cca` calls `permutest.cca`, fills an `anova` table and uses `print.anova` for printing.

Note

The default permutation `model` changed from `"direct"` to `"reduced"` in **vegan** version 1.14-11 (release version 1.15-0), and you must explicitly set `model = "direct"` for compatibility with the old version.

Author(s)

Jari Oksanen

References

Legendre, P. and Legendre, L. (1998). *Numerical Ecology*. 2nd English ed. Elsevier.

See Also

`cca`, `rda`, `capscale` to get something to analyse. Function `drop1.cca` calls `anova.cca` with `by = "margin"`, and `add1.cca` an analysis for single terms additions, which can be used in automatic or semiautomatic model building (see `deviance.cca`.

Examples

```
data(varespec)
data(varechem)
vare.cca <- cca(varespec ~ Al + P + K, varechem)
## overall test
anova(vare.cca)
## Test for axes
anova(vare.cca, by="axis", perm.max=500)
## Sequential test for terms
anova(vare.cca, by="terms", permu=200)
## Marginal or Type III effects
anova(vare.cca, by="margin")
## Marginal test knows 'scope'
anova(vare.cca, by = "m", scope="P")
```

`as.mlm.cca` *Refit Constrained Ordination as a Multiple Response Linear Model*

Description

Functions refit results of constrained ordination (`cca`, `rda`, `capscale`) as a multiple response linear model (`lm`). This allows finding influence statistics (`influence.measures`). This also allows deriving several other statitics, but most of these are biased and misleading, since refitting ignores a major component of variation in constrained ordination.

Usage

```
as.mlm(x)
```

Arguments

`x`	Constrained ordination result.

Details

Popular algorithm for constrained ordination is based on iteration with regression where weighted averages of sites are used as dependent variables and constraints as independent variables. Statistics of linear regression are a natural by-product in this algorithm. Constrained ordination in **vegan** uses different algorithm, but to obtain linear regression statistics you can refit an ordination result as a multiple response linear model (`lm`). This regression ignores residual unconstrained variation in the data, and therefore estimates of standard error are strongly biased and much too low. You can get statistics like t-values of coefficients, but you should not use these because of this bias. Some useful information you can get with refitted models are statistics for detecting influential observations (`influence.measures` including `cooks.distance`, `hatvalues`).

Value

Function returns an object of multiple response linear model of class `"mlm"` documented with `lm`.

Note

You can use these functions to find t-values of coefficients using `summary.mlm`, but you should not do this because the method ignores unconstrained residual variation. You also can find several other statistics for (multiple response) linear models with similar bias. This bias is not a unique feature in **vegan** implementation, but also applies to implementations in other software.

Some statistics of linear models can be found without using these functions: `coef.cca` gives the regression coefficients, `spenvcor` the species-environment correlation, `intersetcor` the interset correlation, `vif.cca` the variance inflation factors.

Author(s)

Jari Oksanen

See Also

cca, rda, capscale, cca.object, lm, summary.mlm, influence.measures.

Examples

```
data(varespec)
data(varechem)
mod <- cca(varespec ~ Al + P + K, data=varechem)
lmod <- as.mlm(mod)
## Coefficients
lmod
coef(mod)
## Influential observations
influence.measures(lmod)
plot(mod, type = "n")
points(mod, cex = 10*hatvalues(lmod), pch=16, xpd = TRUE)
text(mod, display = "bp", col = "blue")
```

---

beals *Beals Smoothing and Degree of Absence*

Description

Beals smoothing replaces each entry in the community data with a probability of target species occurring in that particular site, based on the joint occurrences of target species with the species that actually occur in the site. Swan's (1970) degree of absence applies Beals smoothing to zero items so long that all zeros are replaced with smoothed values.

Usage

```
beals(x)
swan(x)
```

Arguments

x	Community data frame or matrix

Details

Beals smoothing is the estimated probability p_{ij} that species j occurs in site i. It is defined as $p_{ij} = \frac{1}{S_i} \sum_k \frac{N_{jk} I_{ik}}{N_k}$, where S_i is the number of species on site i, N_{jk} is the number of joint occurrences of species j and k, N_k is the number of occurences of species k, and I is the incidence (0 or 1) of species (this last term is usually omitted from the equation, but it is necessary).

Beals smoothing was originally suggested as a method of data transformation to remove excessive zeros (Beals 1984, McCune 1994). However, it is not a suitable method for this purpose since it does not maintain the information on species presences: A species may have a higher probability of occurrence in a site where it does not occur than in sites where it occurs. Moreover, it regularizes data too strongly. The method may be useful in identifying species that belong to the species pool

(Ewald 2002) or to identify suitable unoccupied patches in metapopulation analysis (Münzbergová & Herben 2004). The function is provided here for the brave.

Swan (1970) suggested replacing zero values with degrees of absence of a species in a community data matrix. Swan expressed the method in terms of a similarity matrix, but it is equivalent to applying Beals smoothing to zero values, at each step shifting the smallest initially non-zero item to value one, and repeating this so many times that there are no zeros left in the data. This is actually very similar to extended dissimilarities (implemented in function `stepacross`), but very rarely used.

Value

The function returns a transformed data matrix.

Note

The current function is modelled as closely as possible after Beals (1984) and McCune (1994). It seems that Münzbergová and Herben (2004) use slightly different formulation.

Author(s)

Jari Oksanen

References

Beals, E.W. 1984. Bray-Curtis-ordination: an effective strategy for analysis of multivariate ecological data. *Adv. Ecol. Res.* 14: 1-55.

Ewald, J. 2002. A probabilistic approach to estimating species pools from large compositional matrices. *J. Veg. Sci.* 13: 191-198.

McCune, B. 1994. Improving community ordination with the Beals smoothing function. *Ecoscience* 1: 82-86.

Münzbergová, Z. & Herben, T. 2004. Identification of suitable unoccupied habitats in metapopulation studies using co-occurrence of species. *Oikos* 105: 408-414.

Swan, J.M.A. (1970) An examination of some ordination problems by use of simulated vegetational data. *Ecology* 51, 89–102.

See Also

`decostand` for proper standardization methods, `specpool` for an attempt to assess the size of species pool.

Examples

```
data(dune)
x <- beals(dune)
## Smoothed values against presence or absence of species
pa <- decostand(dune, "pa")
boxplot(as.vector(x) ~ unlist(pa), xlab="Presence", ylab="Beals")
```

`betadisper` *Multivariate homogeneity of groups dispersions (variances)*

Description

Implements Marti Anderson's PERMDISP2 procedure for the analysis of multivariate homogeneity of group dispersions (variances). `betadisper` is a multivariate analogue of Levene's test for homogeneity of variances. Non-euclidean distances between objects and group centroids are handled by reducing the original distances to principal coordinates. This procedure has latterly been used as a means of assessing beta diversity. There are `anova`, `scores`, `plot` and `boxplot` methods.

`TukeyHSD.betadisper` creates a set of confidence intervals on the differences between the mean distance-to-centroid of the levels of the grouping factor with the specified family-wise probability of coverage. The intervals are based on the Studentized range statistic, Tukey's 'Honest Significant Difference' method.

Usage

```
betadisper(d, group, type = c("centroid", "median"))

## S3 method for class 'betadisper':
anova(object, ...)

## S3 method for class 'betadisper':
scores(x, display = c("sites", "centroids"),
       choices = c(1,2), ...)

## S3 method for class 'betadisper':
plot(x, axes = c(1,2), cex = 0.7, hull = TRUE,
     ylab, xlab, main, sub, ...)

## S3 method for class 'betadisper':
boxplot(x, ylab = "Distance to centroid", ...)

## S3 method for class 'betadisper':
TukeyHSD(x, which = "group", ordered = FALSE,
         conf.level = 0.95, ...)
```

Arguments

`d`	a distance structure such as that returned by `dist`, `betadiver` or `vegdist`.
`group`	vector describing the group structure, usually a factor or an object that can be coerced to a factor using `as.factor`. Can consist of a factor with a single level (i.e. one group).
`type`	the type of analysis to perform. Only `type = "centroid"` is currently supported.
`display`	character; partial match to access scores for `"sites"` or `"species"`.

`object, x`	an object of class `"betadisper"`, the result of a call to `betadisper`.
`choices, axes`	the principal coordinate axes wanted.
`hull`	logical; should the convex hull for each group be plotted?
`cex, ylab, xlab, main, sub`	graphical parameters. For details, see `plot.default`.
`which`	A character vector listing terms in the fitted model for which the intervals should be calculated. Defaults to the grouping factor.
`ordered`	Logical; see `TukeyHSD`.
`conf.level`	A numeric value between zero and one giving the family-wise confidence level to use.
`...`	arguments, including graphical parameters (for `plot.betadisper` and `boxplot.betadisper`), passed to other methods.

Details

One measure of multivariate dispersion (variance) for a group of samples is to calculate the average distance of group members to the group centroid or spatial median in multivariate space. To test if the dispersions (variances) of one or more groups are different, the distances of group members to the group centroid are subject to ANOVA. This is a multivariate analogue of Levene's test for homogeneity of variances if the distances between group members and group centroids is the Euclidean distance.

However, better measures of distance than the Euclidean distance are available for ecological data. These can be accommodated by reducing the distances produced using any dissimilarity coefficient to principal coordinates, which embeds them within a Euclidean space. The analysis then proceeds by calculating the Euclidean distances between group members and the group centroid on the basis of the principal coordinate axes rather than the original distances.

Non-metric dissimilarity coefficients can produce principal coordinate axes that have negative Eigenvalues. These correspond to the imaginary, non-metric part of the distance between objects. If negative Eigenvalues are produced, we must correct for these imaginary distances.

The distance to its centroid of a point is

$$z_{ij}^c = \sqrt{\Delta^2(u_{ij}^+, c_i^+) - \Delta^2(u_{ij}^-, c_i^-)},$$

where Δ^2 is the squared Euclidean distance between u_{ij}, the principal coordinate for the j^{th} point in the i^{th} group, and c_i, the coordinate of the centroid for the i^{th} group. The super-scripted $+$ and $-$ indicate the real and imaginary parts respectively. This is equation (3) in Anderson (2006). If the imaginary part is greater in magnitude than the real part, then we would be taking the square root of a negative value, resulting in NaN. From **vegan** 1.12-12 `betadisper` takes the absolute value of the real distance minus the imaginary distance, before computing the square root. This is in line with the behaviour of Marti Anderson's PERMDISP2 programme.

To test if one or more groups is more variable than the others, ANOVA of the distances to group centroids can be performed and parametric theory used to interpret the significance of F. An alternative is to use a permutation test. `permutest.betadisper` permutes model residuals to generate a permutation distribution of F under the Null hypothesis of no difference in dispersion between groups.

Pairwise comprisons of group mean dispersions can also be performed using `permutest.betadisper`. An alternative to the classical comparison of group dispersions, is to calculate Tukey's Honest Significant Differences between groups, via `TukeyHSD.betadisper`. This is a simple wrapper to `TukeyHSD.aov`. The user is directed to read the help file for `TukeyHSD` before using this function. In particular, note the statement about using the function with unbalanced designs.

The results of the analysis can be visualised using the `plot` and `boxplot` methods.

One additional use of these functions is in assessing beta diversity (Anderson *et al* 2006). Function `betadiver` provides some popular dissimilarity measures for this purpose.

Value

The `anova` method returns an object of class `"anova"` inheriting from class `"data.frame"`.

The `scores` method returns a list with one or both of the components `"sites"` and `"centroids"`.

The `plot` function invisibly returns an object of class `"ordiplot"`, a plotting structure which can be used by `identify.ordiplot` (to identify the points) or other functions in the `ordiplot` family.

The `boxplot` function invisibly returns a list whose components are documented in `boxplot`.

`TukeyHSD.betadisper` returns a list. See `TukeyHSD` for further details.

`betadisper` returns a list of class `"betadisper"` with the following components:

`eig`	numeric; the eigenvalues of the principal coordinates analysis.
`vectors`	matrix; the eigenvectors of the principal coordinates analysis.
`distances`	numeric; the Euclidean distances in principal coordinate space between the samples and their respective group centroid.
`group`	factor; vector describing the group structure
`centroids`	matrix; the locations of the group centroids on the principal coordinates.
`call`	the matched function call.

Note

If `group` consists of a single level or group, then the `anova` and `permutest` methods are not appropriate and if used on such data will stop with an error.

Author(s)

Gavin L. Simpson

References

Anderson, M.J. (2006) Distance-based tests for homogeneity of multivariate dispersions. *Biometrics* **62(1)**, 245–253.

Anderson, M.J., Ellingsen, K.E. & McArdle, B.H. (2006) Multivariate dispersion as a measure of beta diversity. *Ecology Letters* **9(6)**, 683–693.

See Also

`permutest.betadisper`, `anova.lm`, `scores`, `boxplot`, `TukeyHSD`. Further measure of beta diversity can be found in `betadiver`.

Examples

```
data(varespec)

## Bray-Curtis distances between samples
dis <- vegdist(varespec)

## First 16 sites grazed, remaining 8 sites ungrazed
groups <- factor(c(rep(1,16), rep(2,8)), labels = c("grazed","ungrazed"))

## Calculate multivariate dispersions
mod <- betadisper(dis, groups)
mod

## Perform test
anova(mod)

## Permutation test for F
permutest(mod, pairwise = TRUE)

## Tukey's Honest Significant Differences
(mod.HSD <- TukeyHSD(mod))
plot(mod.HSD)

## Plot the groups and distances to centroids on the
## first two PCoA axes
plot(mod)

## Draw a boxplot of the distances to centroid for each group
boxplot(mod)
```

`betadiver` *Indices of beta Diversity*

Description

The function estimates any of the 24 indices of beta diversity reviewed by Koleff et al. (2003). Alternatively, it finds the co-occurrence frequencies for triangular plots (Koleff et al. 2003).

Usage

```
betadiver(x, index = NA, order = FALSE, help = FALSE, ...)
## S3 method for class 'betadiver':
plot(x, ...)
## S3 method for class 'betadiver':
scores(x, triangular = TRUE, ...)
```

Arguments

`x`	Community data matrix, or the `betadiver` result for `plot` and `scores` functions.
`index`	The index of beta diversity as defined in Koleff et al. (2003), Table 1. You can use either the subscript of β or the number of the index. See argument `help` below.
`order`	Order sites by increasing number of species. This will influence the configuration in the triangular plot and non-symmetric indices.
`help`	Show the numbers, subscript names and the defining equations of the indices and exit.
`triangular`	Return scores suitable for triangular plotting of proportions. If `FALSE`, returns a 3-column matrix of raw counts.
`...`	Other arguments to functions.

Details

The most commonly used index of beta diversity is $\beta_w = S/\alpha - 1$, where S is the total number of species, and α is the average number of species per site (Whittaker 1960). A drawback of this model is that S increases with sample size, but the expectation of α remains constant, and so the beta diversity increases with sample size. A solution to this problem is to study the beta diversity of pairs of sites. If we denote the number of species shared between two sites as a and the numbers of unique species (not shared) as b and c, then $S = a + b + c$ and $\alpha = (2a + b + c)/2$ so that $\beta_w = (b + c)/(2a + b + c)$. This is the Sørensen dissimilarity as defined in **vegan** function `vegdist` with argument `binary = TRUE`. Many other indices are dissimilarity indices as well.

Function `betadiver` finds all indices reviewed by Koleff et al. (2003). All these indices could be found with function `designdist` which uses different notation, but the current function provides a conventional shortcut. The function only finds the indices. The proper analysis must be done with functions such as `betadisper`, `adonis` or `mantel`.

The indices are directly taken from Table 1 of Koleff et al. (2003), and they can be selected either by the index number or the subscript name used by Koleff et al. The numbers, names and defining equations can be seen using `betadiver(help = TRUE)`. In all cases where there are two alternative forms, the one with the term -1 is used. There are several duplicate indices, and the number of distinct alternatives is much lower than 24 formally provided. The formulations used in functions differ occasionally from those in Koleff et al. (2003), but they are still mathematically equivalent. With `index = NA`, no index is calculated, but instead an object of class `betadiver` is returned. This is a list of elements `a`, `b` and `c`. Function `plot` can be used to display the proportions of these elements in triangular plot as suggested by Koleff et al. (2003), and `scores` extracts the triangular coordinates or the raw scores. Function `plot` returns invisibly the triangular coordinates as an `"ordiplot"` object.

Value

With `index = NA`, the function returns an object of class `"betadisper"` with elements `a`, `b`, and `c`. If `index` is specified, the function returns a `"dist"` object which can be used in any function analysing dissimilarities. For beta diversity, particularly useful functions are `betadisper` to study the betadiversity in groups, `adonis` for any model, and `mantel` to compare beta diversities to other dissimilarities or distances (including geographical distances). Although `betadiver` returns a `"dist"` object, some indices are similarities and cannot be used as such in place of dissimilarities, but that is a severe user error. Functions 10 (`"j"`) and 11 (`"sor"`) are two such similarity indices.

Warning

Some indices return similarities instead of dissimilarities.

Author(s)

Jari Oksanen

References

Koleff, P., Gaston, K.J. and Lennon, J.J. (2003) Measuring beta diversity for presence-absence data. *Journal of Animal Ecology* 72, 367–382.

Whittaker, R.H. (1960) Vegetation of Siskiyou mountains, Oregon and California. *Ecological Monographs* 30, 279–338.

See Also

`designdist` for an alternative to implement all these functions, `vegdist` for some canned alternatives, and `betadisper`, `adonis`, `mantel` for analysing beta diversity objects.

Examples

```
## Raw data and plotting
data(sipoo)
m <- betadiver(sipoo)
plot(m)
## The indices
betadiver(help=TRUE)
## The basic Whittaker index
d <- betadiver(sipoo, "w")
## This should be equal to Sorensen index (binary Bray-Curtis in
## vegan)
range(d - vegdist(sipoo, binary=TRUE))
```

`bgdispersal` *Coefficients of Biogeographical Dispersal Direction*

Description

This function computes coefficients of dispersal direction between geographically connected areas, as defined by Legendre and Legendre (1984), and also described in Legendre and Legendre (1998, section 13.3.4).

Usage

```
bgdispersal(mat, PAonly = FALSE, abc = FALSE)
```

Arguments

`mat`	Data frame or matrix containing a community composition data table (species presence-absence or abundance data).
`PAonly`	`FALSE` if the four types of coefficients, DD1 to DD4, are requested; `TRUE` if `DD1` and `DD2` only are sought (see Details).
`abc`	If `TRUE`, return tables `a`, `b` and `c` used in `DD1` and `DD2`.

Details

The signs of the DD coefficients indicate the direction of dispersal, provided that the asymmetry is significant. A positive sign indicates dispersal from the first (row in DD tables) to the second region (column); a negative sign indicates the opposite. A McNemar test of asymmetry is computed from the presence-absence data to test the hypothesis of a significant asymmetry between the two areas under comparison.

In the input data table, the rows are sites or areas, the columns are taxa. Most often, the taxa are species, but the coefficients can be computed from genera or families as well. DD1 and DD2 only are computed for presence-absence data. The four types of coefficients are computed for quantitative data, which are converted to presence-absence for the computation of DD1 and DD2. `PAonly = FALSE` indicates that the four types of coefficients are requested. `PAonly = TRUE` if DD1 and DD2 only are sought.

Value

Function `bgdispersal` returns a list containing the following matrices:

`DD1`	$DD1[j,k] = (a*(b-c))/((a+b+c)^2)$
`DD2`	$DD2[j,k] = (2*a*(b-c))/((2*a+b+c)*(a+b+c))$ where a, b, and c have the same meaning as in the computation of binary similarity coefficients.
`DD3`	DD3[j,k] = $W*(A-B)/((A+B-W)^2)$
`DD4`	DD4[j,k] = $2*W*(A-B)/((A+B)*(A+B-W))$ where `W = sum(pmin(vector1, vector2))`, `A = sum(vector1)`, `B = sum(vector2)`
`McNemar`	McNemar chi-square statistic of asymmetry (Sokal and Rohlf 1995): $2*(b*log(b)+c*log(c)-(b+c)*log((b+c)/2))/q$ where $q = 1+1/(2*(b+c))$ (Williams correction for continuity)
`prob.McNemar`	probabilities associated with McNemar statistics, chi-square test. H0: no asymmetry in $(b-c)$.

Note

The function uses a more powerful alternative for the McNemar test than the classical formula. The classical formula was constructed in the spririt of Pearson's Chi-square, but the formula in this function was constructed in the spirit of Wilks Chi-square or the G statistic. Function `mcnemar.test` uses the classical formula. The new formula was introduced in **vegan** version 1.10-11, and the older implementations of `bgdispersal` used the classical formula.

Author(s)

Pierre Legendre, Departement de Sciences Biologiques, Universite de Montreal

References

Legendre, P. and V. Legendre. 1984. Postglacial dispersal of freshwater fishes in the Québec peninsula. *Can. J. Fish. Aquat. Sci.* **41**: 1781-1802.

Legendre, P. and L. Legendre. 1998. *Numerical ecology*, 2nd English edition. Elsevier Science BV, Amsterdam.

Sokal, R. R. and F. J. Rohlf. 1995. *Biometry. The principles and practice of statistics in biological research.* 3rd edn. W. H. Freeman, New York.

Examples

```
mat <- matrix(c(32,15,14,10,70,30,100,4,10,30,25,0,18,0,40,
  0,0,20,0,0,0,0,4,0,30,20,0,0,0,0,25,74,42,1,45,89,5,16,16,20),
  4, 10, byrow=TRUE)
bgdispersal(mat)
```

`bioenv` *Best Subset of Environmental Variables with Maximum (Rank) Correlation with Community Dissimilarities*

Description

Function finds the best subset of environmental variables, so that the Euclidean distances of scaled environmental variables have the maximum (rank) correlation with community dissimilarities.

Usage

```
## Default S3 method:
bioenv(comm, env, method = "spearman", index = "bray",
       upto = ncol(env), trace = FALSE, partial = NULL, ...)
## S3 method for class 'formula':
bioenv(formula, data, ...)
```

Arguments

`comm`	Community data frame.
`env`	Data frame of continuous environmental variables.
`method`	The correlation method used in `cor`.
`index`	The dissimilarity index used for community data in `vegdist`.

`upto`	Maximum number of parameters in studied subsets.
`formula, data`	Model `formula` and data.
`trace`	Trace the advance of calculations
`partial`	Dissimilarities partialled out when inspecting variables in `env`.
`...`	Other arguments passed to `cor`.

Details

The function calculates a community dissimilarity matrix using `vegdist`. Then it selects all possible subsets of environmental variables, `scale`s the variables, and calculates Euclidean distances for this subset using `dist`. Then it finds the correlation between community dissimilarities and environmental distances, and for each size of subsets, saves the best result. There are $2^p - 1$ subsets of p variables, and an exhaustive search may take a very, very, very long time (parameter `upto` offers a partial relief).

The function can be called with a model `formula` where the LHS is the data matrix and RHS lists the environmental variables. The formula interface is practical in selecting or transforming environmental variables.

With argument `partial` you can perform "partial" analysis. The partializing item must be a dissimilarity object of class `dist`. The `partial` item can be used with any correlation `method`, but it is strictly correct only for Pearson.

Clarke & Ainsworth (1993) suggested this method to be used for selecting the best subset of environmental variables in interpreting results of nonmetric multidimensional scaling (NMDS). They recommended a parallel display of NMDS of community dissimilarities and NMDS of Euclidean distances from the best subset of scaled environmental variables. They warned against the use of Procrustes analysis, but to me this looks like a good way of comparing these two ordinations.

Clarke & Ainsworth wrote a computer program BIO-ENV giving the name to the current function. Presumably BIO-ENV was later incorporated in Clarke's PRIMER software (available for Windows). In addition, Clarke & Ainsworth suggested a novel method of rank correlation which is not available in the current function.

Value

The function returns an object of class `bioenv` with a `summary` method.

Note

If you want to study the 'significance' of `bioenv` results, you can use function `mantel` or `mantel.partial` which use the same definition of correlation. However, `bioenv` standardizes environmental variables to unit standard deviation using function `scale` and you must do the same in `mantel` for comparable results. Further, `bioenv` selects variables to maximize the Mantel correlation, and significance tests based on *a priori* selection of variables are biased.

Author(s)

Jari Oksanen. The code for selecting all possible subsets was posted to the R mailing list by Prof. B. D. Ripley in 1999.

References

Clarke, K. R & Ainsworth, M. 1993. A method of linking multivariate community structure to environmental variables. *Marine Ecology Progress Series*, 92, 205–219.

See Also

`vegdist`, `dist`, `cor` for underlying routines, `isoMDS` for ordination, `procrustes` for Procrustes analysis, `protest` for an alternative, and `rankindex` for studying alternatives to the default Bray-Curtis index.

Examples

```
# The method is very slow for large number of possible subsets.
# Therefore only 6 variables in this example.
data(varespec)
data(varechem)
sol <- bioenv(wisconsin(varespec) ~ log(N) + P + K + Ca + pH + Al, varechem)
sol
summary(sol)
```

`biplot.rda` *PCA biplot*

Description

Draws a PCA biplot with species scores indicated by biplot arrows

Usage

```
## S3 method for class 'rda':
biplot(x, choices = c(1, 2), scaling = 2,
       display = c("sites", "species"), type, xlim, ylim, col = c(1,2), ...)
```

Arguments

`x`	A `rda` result object.
`choices`	Axes to show.
`scaling`	Scaling for species and site scores. Either species (`2`) or site (`1`) scores are scaled by eigenvalues, and the other set of scores is left unscaled, or with `3` both are scaled symmetrically by square root of eigenvalues. With negative scaling values in `rda`, species scores are divided by standard deviation of each species and multiplied with an equalizing constant. Unscaled raw scores stored in the result can be accessed with `scaling = 0`.
`display`	Scores shown. These must some of the alternatives `"species"` for species scores, and/or `"sites"` for site scores.

`type`	Type of plot: partial match to `text` for text labels, `points` for points, and `none` for setting frames only. If omitted, `text` is selected for smaller data sets, and `points` for larger. Can be of length 2 (e.g. `type = c("text", "points")`), in which case the first element describes how species scores are handled, and the second how site scores are drawn.
`xlim, ylim`	the x and y limits (min, max) of the plot.
`col`	Colours used for sites and species (in this order). If only one colour is given, it is used for both.
`...`	Other parameters for plotting functions.

Details

Produces a plot or biplot of the results of a call to `rda`. It is common for the "species" scores in a PCA to be drawn as biplot arrows that point in the direction of increasing values for that variable. The `biplot.rda` function provides a wrapper to `plot.cca` to allow the easy production of such a plot.

`biplot.rda` is only suitable for unconstrained models. If used on an ordination object with constraints, an error is issued.

If species scores are drawn using `"text"`, the arrows are drawn from the origin to 0.85 * species score, whilst the labels are drawn at the species score. If the type used is `"points"`, then no labels are drawn and therefore the arrows are drawn from the origin to the actual species score.

Value

The `plot` function returns invisibly a plotting structure which can be used by `identify.ordiplot` to identify the points or other functions in the `ordiplot` family.

Author(s)

Gavin Simpson, based on `plot.cca` by Jari Oksanen.

See Also

`plot.cca`, `rda` for something to plot, `ordiplot` for an alternative plotting routine and more support functions, and `text`, `points` and `arrows` for the basic routines.

Examples

```
data(dune)
mod <- rda(dune, scale = TRUE)
biplot(mod, scaling = 3)

## different type for species and site scores
biplot(mod, scaling = 3, type = c("text", "points"))
```

`capscale` *[Partial] Constrained Analysis of Principal Coordinates or distance-based RDA*

Description

Constrained Analysis of Principal Coordinates (CAP) is an ordination method similar to Redundancy Analysis (`rda`), but it allows non-Euclidean dissimilarity indices, such as Manhattan or Bray–Curtis distance. Despite this non-Euclidean feature, the analysis is strictly linear and metric. If called with Euclidean distance, the results are identical to `rda`, but `capscale` will be much more inefficient. Function `capscale` is a constrained version of metric scaling, a.k.a. principal coordinates analysis, which is based on the Euclidean distance but can be used, and is more useful, with other dissimilarity measures. The function can also perform unconstrained principal coordinates analysis, optionally using extended dissimilarities.

Usage

```
capscale(formula, data, distance = "euclidean", comm = NULL,
         add = FALSE,  dfun = vegdist, metaMDSdist = FALSE, ...)
```

Arguments

`formula`	Model formula. The function can be called only with the formula interface. Most usual features of `formula` hold, especially as defined in `cca` and `rda`. The LHS must be either a community data matrix or a dissimilarity matrix, e.g., from `vegdist` or `dist`. If the LHS is a data matrix, function `vegdist` will be used to find the dissimilarities. The RHS defines the constraints. The constraints can be continuous variables or factors, they can be transformed within the formula, and they can have interactions as in a typical `formula`. The RHS can have a special term `Condition` that defines variables to be "partialled out" before constraints, just like in `rda` or `cca`. This allows the use of partial CAP.
`data`	Data frame containing the variables on the right hand side of the model formula.
`distance`	Dissimilarity (or distance) index in `vegdist` used if the LHS of the `formula` is a data frame instead of dissimilarity matrix.
`comm`	Community data frame which will be used for finding species scores when the LHS of the `formula` was a dissimilarity matrix. This is not used if the LHS is a data frame. If this is not supplied, the "species scores" are the axes of initial metric scaling (`cmdscale`) and may be confusing.
`add`	Logical indicating if an additive constant should be computed, and added to the non-diagonal dissimilarities such that all eigenvalues are non-negative in the underlying Principal Co-ordinates Analysis (see `cmdscale` for details). This implements "correction method 2" of Legendre & Legendre (1998, p. 434). The negative eigenvalues are caused by using semi-metric or non-metric dissimilarities with basically metric `cmdscale`. They are harmless and ignored in `capscale`, but you also can avoid warnings with this option.

`dfun`	Distance or dissimilarity function used. Any function returning standard `"dist"` and taking the index name as the first argument can be used.
`metaMDSdist`	Use `metaMDSdist` similarly as in `metaMDS`. This means automatic data transformation and using extended flexible shortest path dissimilarities (function `stepacross`) when there are many dissimilarities based on no shared species.
`...`	Other parameters passed to `rda` or to `metaMDSdist`.

Details

Canonical Analysis of Principal Coordinates (CAP) is simply a Redundancy Analysis of results of Metric (Classical) Multidimensional Scaling (Anderson & Willis 2003). Function capscale uses two steps: (1) it ordinates the dissimilarity matrix using `cmdscale` and (2) analyzes these results using `rda`. If the user supplied a community data frame instead of dissimilarities, the function will find the needed dissimilarity matrix using `vegdist` with specified `distance`. However, the method will accept dissimilarity matrices from `vegdist`, `dist`, or any other method producing similar matrices. The constraining variables can be continuous or factors or both, they can have interaction terms, or they can be transformed in the call. Moreover, there can be a special term `Condition` just like in `rda` and `cca` so that "partial" CAP can be performed.

The current implementation differs from the method suggested by Anderson & Willis (2003) in three major points which actually make it similar to distance-based redundancy analysis (Legendre & Anderson 1999):

1. Anderson & Willis used the orthonormal solution of `cmdscale`, whereas `capscale` uses axes weighted by corresponding eigenvalues, so that the ordination distances are the best approximations of original dissimilarities. In the original method, later "noise" axes are just as important as first major axes.
2. Anderson & Willis take only a subset of axes, whereas `capscale` uses all axes with positive eigenvalues. The use of subset is necessary with orthonormal axes to chop off some "noise", but the use of all axes guarantees that the results are the best approximation of original dissimilarities.
3. Function `capscale` adds species scores as weighted sums of (residual) community matrix (if the matrix is available), whereas Anderson & Willis have no fixed method for adding species scores.

With these definitions, function `capscale` with Euclidean distances will be identical to `rda` in eigenvalues and in site, species and biplot scores (except for possible sign reversal). However, it makes no sense to use `capscale` with Euclidean distances, since direct use of `rda` is much more efficient. Even with non-Euclidean dissimilarities, the rest of the analysis will be metric and linear.

The function can be also used to perform ordinary metric scaling a.k.a. principal coordinates analysis by using a formula with only a constant on the left hand side, or `comm ~ 1`. With `metaMDSdist = TRUE`, the function can do automatic data standardization and use extended dissimilarities using function `stepacross` similary as in non-metric multidimensional scaling with `metaMDS`.

Value

The function returns an object of class `capscale` which is identical to the result of `rda`. At the moment, `capscale` does not have specific methods, but it uses `cca` and `rda` methods `plot.cca`, scores.rda etc. Moreover, you can use anova.cca for permutation tests of “significance” of the results.

Note

Warnings of negative eigenvalues are issued with most dissimilarity indices. These are harmless, and negative eigenvalues will be ignored in the analysis. If the warnings are disturbing, you can use argument `add = TRUE` passed to cmdscale, or, preferably, a distance measure that does not cause these warnings. In vegdist, `method = "jaccard"` gives such an index. Alternatively, after square root transformation many indices do not cause warnings.

Function rda usually divides the ordination scores by number of sites minus one. In this way, the inertia is variance instead of sum of squares, and the eigenvalues sum up to variance. Many dissimilarity measures are in the range 0 to 1, so they have already made a similar division. If the largest original dissimilarity is less than or equal to 4 (allowing for stepacross), this division is undone in `capscale` and original dissimilarities are used. The inertia is named `squared dissimilarity` (as defined in the dissimilarity matrix), but keyword `mean` is added to the inertia in cases where division was made, e.g. in Euclidean and Manhattan distances.

Author(s)

Jari Oksanen

References

Anderson, M.J. & Willis, T.J. (2003). Canonical analysis of principal coordinates: a useful method of constrained ordination for ecology. *Ecology* 84, 511–525.

Legendre, P. & Anderson, M. J. (1999). Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. *Ecological Monographs* 69, 1–24.

Legendre, P. & Legendre, L. (1998). *Numerical Ecology*. 2nd English Edition. Elsevier

See Also

rda, cca, plot.cca, anova.cca, vegdist, dist, cmdscale.

Examples

```
data(varespec)
data(varechem)
vare.cap <- capscale(varespec ~ N + P + K + Condition(Al), varechem,
                     dist="bray")
vare.cap
plot(vare.cap)
anova(vare.cap)
## Principal coordinates analysis with extended dissimilarities
capscale(varespec ~ 1, dist="bray", metaMDS = TRUE)
```

`cascadeKM` *K-means partitioning using a range of values of K*

Description

This function is a wrapper for the `kmeans` function. It creates several partitions forming a cascade from a small to a large number of groups.

Usage

```
cascadeKM(data, inf.gr, sup.gr, iter = 100, criterion = "calinski")

cIndexKM(y, x, index = "all")

## S3 method for class 'cascadeKM':
plot(x, min.g, max.g, grpmts.plot = TRUE,
     sortg = FALSE, gridcol = NA, ...)
```

Arguments

`data`	The data matrix. The objects (samples) are the rows.
`inf.gr`	The number of groups for the partition with the smallest number of groups of the cascade (min).
`sup.gr`	The number of groups for the partition with the largest number of groups of the cascade (max).
`iter`	The number of random starting configurations for each value of K.
`criterion`	The criterion that will be used to select the best partition. The default value is `"calinski"`, which refers to the Calinski-Harabasz (1974) criterion. The simple structure index (`"ssi"`) is also available. Other indices are available in function `clustIndex` (package **cclust**). In our experience, the two indices that work best and are most likely to return their maximum value at or near the optimal number of clusters are `"calinski"` and `"ssi"`.
`y`	Object of class `"kmeans"` returned by a clustering algorithm such as `kmeans`
`x`	Data matrix where columns correspond to variables and rows to observations, or the plotting object in `plot`
`index`	The available indices are: `"calinski"` and `"ssi"`. Type `"all"` to obtain both indices. Abbreviations of these names are also accepted.
`min.g, max.g`	The minimum and maximum numbers of groups to be displayed.
`grpmts.plot`	Show the plot (`TRUE` or `FALSE`).
`sortg`	Sort the objects as a function of their group membership to produce a more easily interpretatable graph. See Details. The original object names are kept; they are used as labels in the output table `x`, although not in the graph. If there were no rownames, sequential row numbers are used to keep track of the original order of the objects.

`gridcol`	The colour of the grid lines in the plots. `NA`, which is the default value, removes the grid lines.
`...`	Other parameters to the functions (ignored).

Details

The function creates several partitions formimg a cascade from a small to a large number of groups formed by `kmeans`. Most of the work is performed by function `cIndex` which is based on the `clustIndex` function (package **cclust**). Some of the criteria were removed from this version because computation errors were generated when only one object was found in a group.

The default value is `"calinski"`, which refers to the well-known Calinski-Harabasz (1974) criterion. The other available index is the simple structure index `"ssi"` (Dolnicar et al. 1999). In the case of groups of equal sizes, `"calinski"` is generally a good criterion to indicate the correct number of groups. Users should not take its indications literally when the groups are not equal in size. Type `"all"` to obtain both indices. The indices are defined as:

calinski: $(SSB/(K-1))/(SSW/(n-K))$, where n is the number of data points and K is the number of clusters. SSW is the sum of squares within the clusters while SSB is the sum of squares among the clusters. This index is simply an F (ANOVA) statistic.

ssi: the “Simple Structure Index” multiplicatively combines several elements which influence the interpretability of a partitioning solution. The best partition is indicated by the highest SSI value.

In a simulation study, Milligan and Cooper (1985) found that the Calinski-Harabasz criterion recovered the correct number of groups the most often. We recommend this criterion because, if the groups are of equal sizes, the maximum value of `"calinski"` usually indicates the correct number of groups. Another available index is the simple structure index `"ssi"`. Users should not take the indications of these indices literally when the groups are not equal in size and explore the groups corresponding to other values of K.

Function `cascadeKM` has a `plot` method. Two plots are produced. The graph on the left has the objects in abscissa and the number of groups in ordinate. The groups are represented by colours. The graph on the right shows the values of the criterion (`"calinski"` or `"ssi"`) for determining the best partition. The highest value of the criterion is marked in red. Points marked in orange, if any, indicate partitions producing an increase in the criterion value as the number of groups increases; they may represent other interesting partitions.

If `sortg=TRUE`, the objects are reordered by the following procedure: (1) a simple matching distance matrix is computed among the objects, based on the table of K-means assignments to groups, from K = `min.g` to K = `max.g`. (2) A principal coordinate analysis (PCoA, Gower 1966) is computed on the centred distance matrix. (3) The first principal coordinate is used as the new order of the objects in the graph. A simplified algorithm is used to compute the first principal coordinate only, using the iterative algorithm described in Legendre & Legendre (1998, Table 9.10). The full distance matrix among objects is never computed; this avoids the problem of storing it when the number of objects is large. Distance values are computed as they are needed by the algorithm.

Value

Function `cascadeKM` returns an object of class `cascadeKM` with items:

`partition`	Table with the partitions found for different numbers of groups K, from K = `inf.gr` to K = `sup.gr`.
`results`	Values of the criterion to select the best partition.
`criterion`	The name of the criterion used.
`size`	The number of objects found in each group, for all partitions (columns).

Function `cIndex` returns a vector with the index values. The maximum value of these indices is supposed to indicate the best partition. These indices work best with groups of equal sizes. When the groups are not of equal sizes, one should not put too much faith in the maximum of these indices, and also explore the groups corresponding to other values of K.

Author(s)

Marie-Helene Ouellette ⟨Marie-Helene.Ouellette@UMontreal.ca⟩, Sebastien Durand ⟨Sebastien.Durand@UMontreal.ca⟩ and Pierre Legendre ⟨Pierre.Legendre@UMontreal.ca⟩. Edited for **vegan** by Jari Oksanen.

References

Calinski, T. and J. Harabasz. 1974. A dendrite method for cluster analysis. *Commun. Stat.* **3**: 1-27.

Dolnicar, S., K. Grabler and J. A. Mazanec. 1999. A tale of three cities: perceptual charting for analyzing destination images. Pp. 39-62 in: Woodside, A. et al. [eds.] *Consumer psychology of tourism, hospitality and leisure*. CAB International, New York.

Gower, J. C. 1966. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika* **53**: 325-338.

Legendre, P. & L. Legendre. 1998. *Numerical ecology*, 2nd English edition. Elsevier Science BV, Amsterdam.

Milligan, G. W. & M. C. Cooper. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika* **50**: 159-179.

Weingessel, A., Dimitriadou, A. and Dolnicar, S. *An Examination Of Indexes For Determining The Number Of Clusters In Binary Data Sets*, http://www.wu-wien.ac.at/am/wp99.htm#29

See Also

`kmeans`, `clustIndex`.

Examples

```
# Partitioning a (10 x 10) data matrix of random numbers
mat <- matrix(runif(100),10,10)
res <- cascadeKM(mat, 2, 5, iter = 25, criterion = 'calinski')
toto <- plot(res)

# Partitioning an autocorrelated time series
vec <- sort(matrix(runif(30),30,1))
res <- cascadeKM(vec, 2, 5, iter = 25, criterion = 'calinski')
toto <- plot(res)
```

```
# Partitioning a large autocorrelated time series
# Note that we remove the grid lines
vec <- sort(matrix(runif(1000),1000,1))
res <- cascadeKM(vec, 2, 7, iter = 10, criterion = 'calinski')
toto <- plot(res, gridcol=NA)
```

---

cca *[Partial] [Constrained] Correspondence Analysis and Redundancy Analysis*

Description

Function `cca` performs correspondence analysis, or optionally constrained correspondence analysis (a.k.a. canonical correspondence analysis), or optionally partial constrained correspondence analysis. Function `rda` performs redundancy analysis, or optionally principal components analysis. These are all very popular ordination techniques in community ecology.

Usage

```
## S3 method for class 'formula':
cca(formula, data, ...)
## Default S3 method:
cca(X, Y, Z, ...)
## S3 method for class 'formula':
rda(formula, data, scale=FALSE, ...)
## Default S3 method:
rda(X, Y, Z, scale=FALSE, ...)
```

Arguments

`formula`	Model formula, where the left hand side gives the community data matrix, right hand side gives the constraining variables, and conditioning variables can be given within a special function `Condition`.
`data`	Data frame containing the variables on the right hand side of the model formula.
`X`	Community data matrix.
`Y`	Constraining matrix, typically of environmental variables. Can be missing.
`Z`	Conditioning matrix, the effect of which is removed (‘partialled out’) before next step. Can be missing.
`scale`	Scale species to unit variance (like correlations).
`...`	Other arguments for `print` or `plot` functions (ignored in other functions).

Details

Since their introduction (ter Braak 1986), constrained, or canonical, correspondence analysis and its spin-off, redundancy analysis, have been the most popular ordination methods in community ecology. Functions `cca` and `rda` are similar to popular proprietary software `Canoco`, although the implementation is completely different. The functions are based on Legendre & Legendre's (1998) algorithm: in `cca` Chi-square transformed data matrix is subjected to weighted linear regression on constraining variables, and the fitted values are submitted to correspondence analysis performed via singular value decomposition (`svd`). Function `rda` is similar, but uses ordinary, unweighted linear regression and unweighted SVD.

The functions can be called either with matrix-like entries for community data and constraints, or with formula interface. In general, the formula interface is preferred, because it allows a better control of the model and allows factor constraints.

In the following sections, `X`, `Y` and `Z`, although referred to as matrices, are more commonly data frames.

In the matrix interface, the community data matrix `X` must be given, but the other data matrices may be omitted, and the corresponding stage of analysis is skipped. If matrix `Z` is supplied, its effects are removed from the community matrix, and the residual matrix is submitted to the next stage. This is called 'partial' correspondence or redundancy analysis. If matrix `Y` is supplied, it is used to constrain the ordination, resulting in constrained or canonical correspondence analysis, or redundancy analysis. Finally, the residual is submitted to ordinary correspondence analysis (or principal components analysis). If both matrices `Z` and `Y` are missing, the data matrix is analysed by ordinary correspondence analysis (or principal components analysis).

Instead of separate matrices, the model can be defined using a model `formula`. The left hand side must be the community data matrix (`X`). The right hand side defines the constraining model. The constraints can contain ordered or unordered factors, interactions among variables and functions of variables. The defined `contrasts` are honoured in `factor` variables. The constraints can also be matrices (but not data frames). The formula can include a special term `Condition` for conditioning variables ("covariables") "partialled out" before analysis. So the following commands are equivalent: `cca(X, Y, Z)`, `cca(X ~ Y + Condition(Z))`, where `Y` and `Z` refer to constraints and conditions matrices respectively.

Constrained correspondence analysis is indeed a constrained method: CCA does not try to display all variation in the data, but only the part that can be explained by the used constraints. Consequently, the results are strongly dependent on the set of constraints and their transformations or interactions among the constraints. The shotgun method is to use all environmental variables as constraints. However, such exploratory problems are better analysed with unconstrained methods such as correspondence analysis (`decorana`, `ca`) or non-metric multidimensional scaling (`isoMDS`) and environmental interpretation after analysis (`envfit`, `ordisurf`). CCA is a good choice if the user has clear and strong *a priori* hypotheses on constraints and is not interested in the major structure in the data set.

CCA is able to correct the curve artefact commonly found in correspondence analysis by forcing the configuration into linear constraints. However, the curve artefact can be avoided only with a low number of constraints that do not have a curvilinear relation with each other. The curve can reappear even with two badly chosen constraints or a single factor. Although the formula interface makes easy to include polynomial or interaction terms, such terms often produce curved artefacts (that are difficult to interpret), these should probably be avoided.

According to folklore, `rda` should be used with "short gradients" rather than `cca`. However, this is not based on research which finds methods based on Euclidean metric as uniformly weaker than those based on Chi-squared metric. However, standardized Euclidean distance may be an appropriate measures (see Hellinger standardization in `decostand` in particular).

Partial CCA (pCCA; or alternatively partial RDA) can be used to remove the effect of some conditioning or "background" or "random" variables or "covariables" before CCA proper. In fact, pCCA compares models `cca(X ~ Z)` and `cca(X ~ Y + Z)` and attributes their difference to the effect of `Y` cleansed of the effect of `Z`. Some people have used the method for extracting "components of variance" in CCA. However, if the effect of variables together is stronger than sum of both separately, this can increase total Chi-square after "partialling out" some variation, and give negative "components of variance". In general, such components of "variance" are not to be trusted due to interactions between two sets of variables.

The functions have `summary` and `plot` methods which are documented separately (see `plot.cca`, `summary.cca`).

Value

Function `cca` returns a huge object of class `cca`, which is described separately in `cca.object`.

Function `rda` returns an object of class `rda` which inherits from class `cca` and is described in `cca.object`. The scaling used in `rda` scores is described in a separate vignette with this package.

Author(s)

The responsible author was Jari Oksanen, but the code borrows heavily from Dave Roberts (`http://labdsv.nr.usu.edu/`).

References

The original method was by ter Braak, but the current implementations follows Legendre and Legendre.

Legendre, P. and Legendre, L. (1998) *Numerical Ecology*. 2nd English ed. Elsevier.

McCune, B. (1997) Influence of noisy environmental data on canonical correspondence analysis. *Ecology* **78**, 2617-2623.

Palmer, M. W. (1993) Putting things in even better order: The advantages of canonical correspondence analysis. *Ecology* **74**,2215-2230.

Ter Braak, C. J. F. (1986) Canonical Correspondence Analysis: a new eigenvector technique for multivariate direct gradient analysis. *Ecology* **67**, 1167-1179.

See Also

There is a special documentation for `plot.cca` and `summary.cca` functions with their helper functions (`text.cca`, `points.cca`, `scores.cca`). Function `anova.cca` provides an ANOVA like permutation test for the "significance" of constraints. Automatic model building (dangerous!) is discussed in `deviance.cca`. Diagnostic tools, prediction and adding new points in ordination are discussed in `goodness.cca` and `predict.cca`. Function `cca` (library **ade4**) provide alternative implementations of CCA (these are internally quite different). Function `capscale` is a non-Euclidean generalization of `rda`. The result object is described in `cca.object`. You can use as.mlm to refit ordination result as a multiple response linear model to find some descriptive statistics. Design decisions are explained in vignette 'decision-vegan' which also can be accessed with vegandocs.

Examples

```
data(varespec)
data(varechem)
## Common but bad way: use all variables you happen to have in your
## environmental data matrix
vare.cca <- cca(varespec, varechem)
vare.cca
plot(vare.cca)
## Formula interface and a better model
vare.cca <- cca(varespec ~ Al + P*(K + Baresoil), data=varechem)
vare.cca
plot(vare.cca)
## `Partialling out' and `negative components of variance'
cca(varespec ~ Ca, varechem)
cca(varespec ~ Ca + Condition(pH), varechem)
## RDA
data(dune)
data(dune.env)
dune.Manure <- rda(dune ~ Manure, dune.env)
plot(dune.Manure)
## For further documentation:
## Not run:
vegandocs("decision")
## End(Not run)
```

---

cca.object *Result Object from Constrained Ordination with cca, rda or capscale*

Description

Ordination methods cca, rda and capscale return similar result objects. Function capscale inherits from rda and rda inherits from cca. This inheritance structure is due to historic reasons: cca was the first of these implemented in vegan. Hence the nomenclature in cca.object reflects cca. This help page describes the internal structure of the cca object for programmers.

Value

A cca object has the following elements:

call	the function call.
colsum, rowsum	Column and row sums in cca. In rda, item colsum contains standard deviations of species and rowsum is NA.
grand.total	Grand total of community data in cca and NA in rda.

`inertia`	Text used as the name of inertia.
`method`	Text used as the name of the ordination method.
`terms`	The `terms` component of the `formula`. This is missing if the ordination was not called with `formula`.
`terminfo`	Further information on terms with three subitems: `terms` which is like the `terms` component above, but lists conditions and constrainst similarly; `xlev` which lists the factor levels, and `ordered` which is `TRUE` to ordered factors. This is produced by **vegan** internal function `ordiTerminfo`, and it is needed in `predict.cca` with `newdata`. This is missing if the ordination was not called with `formula`.
`tot.chi`	Total inertia or the sum of all eigenvalues.
`pCCA, CCA, CA`	Actual ordination results for conditioned (partial), constrained and unconstrained components of the model. Any of these can be `NULL` if there is no corresponding component. Items `pCCA`, `CCA` and `CA` have similar structure, and contain following items:
`alias`	The names of the aliased constraints or conditions. Function `alias.cca` does not access this item directly, but it finds the aliased variables and their defining equations from the `QR` item.
`biplot`	Biplot scores of constraints. Only in `CCA`.
centroids	(Weighted) centroids of factor levels of constraints. Only in `CCA`. Missing if the ordination was not called with `formula`.
`eig`	Eigenvalues of axes. In `CCA` and `CA`.
`envcentre`	(Weighted) means of the original constraining or conditioning variables. In `pCCA` and in `CCA`.
`Fit`	The fitted values of standardized data matrix after fitting conditions. Only in `pCCA`.
`QR`	The QR decomposition of explanatory variables as produced by `qr`. The constrained ordination algorithm is based on QR decomposition of constraints and conditions (environmental data). The environmental data are first centred in `rda` or weighted and centred in `cca`. The QR decomposition is used in many functions that access `cca` results, and it can be used to find many items that are not directly stored in the object. For examples, see `coef.cca`, `coef.rda`, `vif.cca`, `permutest.cca`, `predict.cca`, `predict.rda`, `calibrate.cca`. For possible uses of this component, see `qr`. In `pCCA` and `CCA`.
`rank`	The rank of the ordination component.
`qrank`	The rank of the constraints which is the difference of the ranks of QR decompositions in `pCCA` and `CCA` components. Only in `CCA`.
`tot.chi`	Total inertia or the sum of all eigenvalues of the component.
`u`	(Weighted) orthonormal site scores. Please note that scaled scores are not stored in the `cca` object, but they are made when the object is accessed with functions like `scores.cca`, `summary.cca` or `plot.cca`, or their `rda` variants. Only in `CCA` and `CA`. In the `CCA` component these are the so-called linear combination scores.

`u.eig`	`u` scaled by eigenvalues. There is no guarantee that any `.eig` variants of scores will be kept in the future releases.
`v`	(Weighted) orthonormal species scores. If missing species were omitted from the analysis, this will contain attribute `na.action` that lists the omitted species. Only in `CCA` and `CA`.
`v.eig`	`v` weighted by eigenvalues.
`wa`	Site scores found as weighted averages (`cca`) or weighted sums (`rda`) of `v` with weights `Xbar`, but the multiplying effect of eigenvalues removed. These often are known as WA scores in `cca`. Only in `CCA`.
`wa.eig`	The direct result of weighted avaraging or weighted summation (matrix multiplication) with the resulting eigenvalue inflation.
`Xbar`	The standardized data matrix after previous stages of analysis. In `CCA` this is after possible `pCCA` or after partialling out the effects of conditions, and in `CA` after both `pCCA` and `CCA`. In `cca` the standardization is Chi-square, and in `rda` centring and optional scaling by species standard deviations using function `scale`.

Author(s)

Jari Oksanen

References

Legendre, P. and Legendre, L. (1998) *Numerical Ecology*. 2nd English ed. Elsevier.

See Also

The description here provides a hacker's interface. For more user friendly acces to the `cca` object see `alias.cca`, `coef.cca`, `deviance.cca`, `predict.cca`, `scores.cca`, `summary.cca`, `vif.cca`, `weights.cca`, `spenvcor` or `rda` variants of these functions. You can use `as.mlm` to cast a `cca.object` into result of multiple response linear model (`lm`) in order to more easily find some statistics (which in principle could be directly found from the `cca.object` as well).

Examples

```
# Some species will be missing in the analysis, because only a subset
# of sites is used below.
data(dune)
data(dune.env)
mod <- cca(dune[1:15,] ~ ., dune.env[1:15,])
# Look at the names of missing species
attr(mod$CCA$v, "na.action")
# Look at the names of the aliased variables:
mod$CCA$alias
# Access directly constrained weighted orthonormal species and site
# scores, constrained eigenvalues and margin sums.
spec <- mod$CCA$v
sites <- mod$CCA$u
eig <- mod$CCA$eig
```

```
rsum <- mod$rowsum
csum <- mod$colsum
```

---

decorana *Detrended Correspondence Analysis and Basic Reciprocal Averaging*

Description

Performs detrended correspondence analysis and basic reciprocal averaging or orthogonal correspondence analysis.

Usage

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

## S3 method for class 'decorana':
plot(x, choices=c(1,2), origin=TRUE,
     display=c("both","sites","species","none"),
     cex = 0.8, cols = c(1,2), type, xlim, ylim, ...)

## S3 method for class 'decorana':
text(x, display = c("sites", "species"), labels,
     choices = 1:2, origin = TRUE, select,  ...)

## S3 method for class 'decorana':
points(x, display = c("sites", "species"),
       choices=1:2, origin = TRUE, select, ...)

## S3 method for class 'decorana':
summary(object, digits=3, origin=TRUE,
        display=c("both", "species","sites","none"), ...)

## S3 method for class 'summary.decorana':
print(x, head = NA, tail = head, ...)

downweight(veg, fraction = 5)

## S3 method for class 'decorana':
scores(x, display=c("sites","species"), choices=1:4,
       origin=TRUE, ...)
```

Arguments

`veg`	Community data, a matrix-like object.
`iweigh`	Downweighting of rare species (0: no).
`iresc`	Number of rescaling cycles (0: no rescaling).

`ira`	Type of analysis (0: detrended, 1: basic reciprocal averaging).
`mk`	Number of segments in rescaling.
`short`	Shortest gradient to be rescaled.
`before`	Hill's piecewise transformation: values before transformation.
`after`	Hill's piecewise transformation: values after transformation – these must correspond to values in `before`.
`x, object`	A `decorana` result object.
`choices`	Axes shown.
`origin`	Use true origin even in detrended correspondence analysis.
`display`	Display only sites, only species, both or neither.
`cex`	Plot character size.
`cols`	Colours used for sites and species.
`type`	Type of plots, partial match to `"text"`, `"points"` or `"none"`.
`labels`	Optional text to be used instead of row names.
`select`	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices of displayed items.
`xlim, ylim`	the x and y limits (min,max) of the plot.
`digits`	Number of digits in summary output.
`head, tail`	Number of rows printed from the head and tail of species and site scores. Default `NA` prints all.
`fraction`	Abundance fraction where downweighting begins.
`...`	Other arguments for `plot` function.

Details

In late 1970s, correspondence analysis became the method of choice for ordination in vegetation science, since it seemed better able to cope with non-linear species responses than principal components analysis. However, even correspondence analysis can produce an arc-shaped configuration of a single gradient. Mark Hill developed detrended correspondence analysis to correct two assumed 'faults' in correspondence analysis: curvature of straight gradients and packing of sites at the ends of the gradient.

The curvature is removed by replacing the orthogonalization of axes with detrending. In orthogonalization successive axes are made non-correlated, but detrending should remove all systematic dependence between axes. Detrending is performed using a five-segment smoothing window with weights (1,2,3,2,1) on `mk` segments — which indeed is more robust than the suggested alternative of detrending by polynomials. The packing of sites at the ends of the gradient is undone by rescaling the axes after extraction. After rescaling, the axis is supposed to be scaled by 'SD' units, so that the average width of Gaussian species responses is supposed to be one over whole axis. Other innovations were the piecewise linear transformation of species abundances and downweighting of rare species which were regarded to have an unduly high influence on ordination axes.

It seems that detrending actually works by twisting the ordination space, so that the results look non-curved in two-dimensional projections ('lolly paper effect'). As a result, the points usually have an easily recognized triangular or diamond shaped pattern, obviously an artefact of detrendingt. Rescaling works differently than commonly presented, too. `decorana` does not use, or even evaluate, the widths of species responses. Instead, it tries to equalize the weighted variance of species scores on axis segments (parameter `mk` has only a small effect, since `decorana` finds the segment number from the current estimate of axis length). This equalizes response widths only for the idealized species packing model, where all species initially have unit width responses and equally spaced modes.

The `summary` method prints the ordination scores, possible prior weights used in downweighting, and the marginal totals after applying these weights. The `plot` method plots species and site scores. Classical `decorana` scaled the axes so that smallest site score was 0 (and smallest species score was negative), but `summary`, `plot` and `scores` use the true origin, unless `origin = FALSE`.

In addition to proper eigenvalues, the function also reports 'decorana values' in detrended analysis. These 'decorana values' are the values that the legacy code of `decorana` returns as 'eigenvalues'. They are estimated internally during iteration, and it seems that detrending interferes the estimation so that these values are generally too low and have unclear interpretation. Moreover, 'decorana values' are estimated before rescaling which will change the eigenvalues. The proper eigenvalues are estimated after extraction of the axes and they are the ratio of biased weighted variances of site and species scores even in detrended and rescaled solutions. The 'decorana values' are provided only for the the compatibility with legacy software, and they should not be used.

Value

`decorana` returns an object of class `"decorana"`, which has `print`, `summary` and `plot` methods.

Note

`decorana` uses the central numerical engine of the original Fortran code (which is in the public domain), or about 1/3 of the original program. I have tried to implement the original behaviour, although a great part of preparatory steps were written in R language, and may differ somewhat from the original code. However, well-known bugs are corrected and strict criteria used (Oksanen & Minchin 1997).

Please note that there really is no need for piecewise transformation or even downweighting within `decorana`, since there are more powerful and extensive alternatives in R, but these options are included for compliance with the original software. If a different fraction of abundance is needed in downweighting, function `downweight` must be applied before `decorana`. Function `downweight` indeed can be applied prior to correspondence analysis, and so it can be used together with `cca`, too.

The function finds only four axes: this is not easily changed.

Author(s)

Mark O. Hill wrote the original Fortran code, the R port was by Jari Oksanen.

References

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

Oksanen, J. and Minchin, P.R. (1997). Instability of ordination results under changes in input data order: explanations and remedies. *Journal of Vegetation Science* **8**, 447–454.

See Also

For unconstrained ordination, non-metric multidimensional scaling in `isoMDS` may be more robust (see also `metaMDS`). Constrained (or 'canonical') correspondence analysis can be made with `cca`. Orthogonal correspondence analysis can be made with `ca`, or with `decorana` or `cca`, but the scaling of results vary (and the one in `decorana` correspondes to `scaling = -1` in `cca`.). See `predict.decorana` for adding new points to an ordination.

Examples

```
data(varespec)
vare.dca <- decorana(varespec)
vare.dca
summary(vare.dca)
plot(vare.dca)

### the detrending rationale:
gaussresp <- function(x,u) exp(-(x-u)^2/2)
x <- seq(0,6,length=15) ## The gradient
u <- seq(-2,8,len=23)   ## The optima
pack <- outer(x,u,gaussresp)
matplot(x, pack, type="l", main="Species packing")
opar <- par(mfrow=c(2,2))
plot(scores(prcomp(pack)), asp=1, type="b", main="PCA")
plot(scores(decorana(pack, ira=1)), asp=1, type="b", main="CA")
plot(scores(decorana(pack)), asp=1, type="b", main="DCA")
plot(scores(cca(pack ~ x), dis="sites"), asp=1, type="b", main="CCA")

### Let's add some noise:
noisy <- (0.5 + runif(length(pack)))*pack
par(mfrow=c(2,1))
matplot(x, pack, type="l", main="Ideal model")
matplot(x, noisy, type="l", main="Noisy model")
par(mfrow=c(2,2))
plot(scores(prcomp(noisy)), type="b", main="PCA", asp=1)
plot(scores(decorana(noisy, ira=1)), type="b", main="CA", asp=1)
plot(scores(decorana(noisy)), type="b", main="DCA", asp=1)
plot(scores(cca(noisy ~ x), dis="sites"), asp=1, type="b", main="CCA")
par(opar)
```

`decostand` *Standardization Methods for Community Ecology*

Description

The function provides some popular (and effective) standardization methods for community ecologists.

Usage

```
decostand(x, method, MARGIN, range.global, na.rm=FALSE)

wisconsin(x)
```

Arguments

`x`	Community data, a matrix-like object.
`method`	Standardization method. See Details for available options.
`MARGIN`	Margin, if default is not acceptable. `1` = rows, and `2` = columns of `x`.
`range.global`	Matrix from which the range is found in `method = "range"`. This allows using same ranges across subsets of data. The dimensions of `MARGIN` must match with `x`.
`na.rm`	Ignore missing values in row or column standardizations.

Details

The function offers following standardization methods for community data:

- `total`: divide by margin total (default `MARGIN = 1`).
- `max`: divide by margin maximum (default `MARGIN = 2`).
- `freq`: divide by margin maximum and multiply by the number of non-zero items, so that the average of non-zero entries is one (Oksanen 1983; default `MARGIN = 2`).
- `normalize`: make margin sum of squares equal to one (default `MARGIN = 1`).
- `range`: standardize values into range $0 \ldots 1$ (default `MARGIN = 2`). If all values are constant, they will be transformed to 0.
- `standardize`: scale `x` to zero mean and unit variance (default `MARGIN = 2`).
- `pa`: scale `x` to presence/absence scale (0/1).
- `chi.square`: divide by row sums and square root of column sums, and adjust for square root of matrix total (Legendre & Gallagher 2001). When used with the Euclidean distance, the distances should be similar to the the Chi-square distance used in correspondence analysis. However, the results from `cmdscale` would still differ, since CA is a weighted ordination method (default `MARGIN = 1`).
- `hellinger`: square root of `method = "total"` (Legendre & Gallagher 2001).

Standardization, as contrasted to transformation, means that the entries are transformed relative to other entries.

All methods have a default margin. `MARGIN=1` means rows (sites in a normal data set) and `MARGIN=2` means columns (species in a normal data set).

Command `wisconsin` is a shortcut to common Wisconsin double standardization where species (`MARGIN=2`) are first standardized by maxima (`max`) and then sites (`MARGIN=1`) by site totals (`tot`).

Most standardization methods will give nonsense results with negative data entries that normally should not occur in the community data. If there are empty sites or species (or constant with `method = "range"`), many standardization will change these into `NaN`.

Value

Returns the standardized data frame, and adds an attribute `"decostand"` giving the name of applied standardization `"method"`.

Note

Common transformations can be made with standard R functions.

Author(s)

Jari Oksanen

References

Legendre, P. & Gallagher, E.D. (2001) Ecologically meaningful transformations for ordination of species data. *Oecologia* **129**; 271–280.

Oksanen, J. (1983) Ordination of boreal heath-like vegetation with principal component analysis, correspondence analysis and multidimensional scaling. *Vegetatio* **52**; 181–189.

Examples

```
data(varespec)
sptrans <- decostand(varespec, "max")
apply(sptrans, 2, max)
sptrans <- wisconsin(varespec)

## Chi-square: PCA similar but not identical to CA.
## Use wcmdscale for weighted analysis and identical results.
sptrans <- decostand(varespec, "chi.square")
plot(procrustes(rda(sptrans), cca(varespec)))
```

`designdist` *Design your own Dissimilarities*

Description

You can define your own dissimilarities using terms for shared and total quantities, number of rows and number of columns. The shared and total quantities can be binary, quadratic or minimum terms. In binary terms, the shared component is number of shared species, and totals are numbers of species on sites. The quadratic terms are cross-products and sums of squares, and minimum terms are sums of parallel minima and row totals.

Usage

```
designdist(x, method = "(A+B-2*J)/(A+B)",
           terms = c("binary", "quadratic", "minimum"),
           abcd = FALSE, name)
```

Arguments

`x`	Input data.
`method`	Equation for your dissimilarities. This can use terms `J` for shared quantity, `A` and `B` for totals, `N` for the number of rows (sites) and `P` for the number of columns (species). The equation can also contain any R functions that accepts vector arguments and returns vectors of the same length.
`terms`	How shared and total components are found. 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 (shared number of species, and number of species for each row).
`abcd`	Use 2x2 contingency table notation for binary data: a is the number of shared species, b and c are the numbers of species occurring only one of the sites but not in both, and d is the number of species that occur on neither of the sites.
`name`	The name you want to use for your index. The default is to combine the `method` equation and `terms` argument.

Details

Most popular dissimilarity measures in ecology can be expressed with the help of terms `J`, `A` and `B`, and some also involve matrix dimensions `N` and `P`. Some examples you can define in `designdist` are:

`A+B-2*J`	`"quadratic"`	squared Euclidean
`A+B-2*J`	`"minimum"`	Manhattan
`(A+B-2*J)/(A+B)`	`"minimum"`	Bray-Curtis
`(A+B-2*J)/(A+B)`	`"binary"`	Sørensen
`(A+B-2*J)/(A+B-J)`	`"binary"`	Jaccard
`(A+B-2*J)/(A+B-J)`	`"minimum"`	Ružička
`(A+B-2*J)/(A+B-J)`	`"quadratic"`	(dis)similarity ratio
`1-J/sqrt(A*B)`	`"binary"`	Ochiai
`1-J/sqrt(A*B)`	`"quadratic"`	cosine complement
`1-phyper(J-1, A, P-A, B)`	`"binary"`	Raup-Crick

The function `designdist` can implement most dissimilarity indices in `vegdist` or elsewhere, and it can also be used to implement many other indices, amongst them, most of those described in Legendre & Legendre (1998). It can also be used to implement all indices of beta diversity described in Koleff et al. (2003), but there also is a specific function `betadiver` for the purpose.

If you want to implement binary dissimilarities based on the 2x2 contingency table notation, you can set `abcd = TRUE`. In this notation `a = J`, `b = A-J`, `c = B-J`, `d = P-A-B+J`. This notation is often used instead fo the more more tangible default notation for reasons that are opaque to me.

Value

`designdist` returns an object of class `dist`.

Note

`designdist` does not use compiled code, and may be slow or use plenty of memory in large data sets. It is very easy to make errors when defining a function by hand. If an index is available in a function using compiled code, it is better to use the canned alternative.

Author(s)

Jari Oksanen

References

Koleff, P., Gaston, K.J. and Lennon, J.J. (2003) Measuring beta diversity for presence–absence data. *J. Animal Ecol.* **72**, 367–382.

Legendre, P. and Legendre, L. (1998) *Numerical Ecology*. 2nd English ed. Elsevier

See Also

`vegdist`, `betadiver`, `dist`.

Examples

```
## Arrhenius dissimilarity: the value of z in the species-area model
## S = c*A^z when combining two sites of equal areas, where S is the
## number of species, A is the area, and c and z are model parameters.
## The A below is not the area (which cancels out), but number of
## species in one of the sites, as defined in designdist().
data(BCI)
dis <- designdist(BCI, "(log(A+B-J)-log(A+B)+log(2))/log(2)")
## This can be used in clustering or ordination...
ordiplot(cmdscale(dis))
## ... or in analysing beta diversity (without gradients)
summary(dis)
```

`deviance.cca` *Statistics Resembling Deviance and AIC for Constrained Ordination*

Description

The functions extract statistics that resemble deviance and AIC from the result of constrained correspondence analysis `cca` or redundancy analysis `rda`. These functions are rarely needed directly, but they are called by `step` in automatic model building. Actually, `cca` and `rda` do not have `AIC` and these functions are certainly wrong.

Usage

```
## S3 method for class 'cca':
deviance(object, ...)

## S3 method for class 'cca':
extractAIC(fit, scale = 0, k = 2, ...)
```

Arguments

`object`	the result of a constrained ordination (`cca` or `rda`).
`fit`	fitted model from constrained ordination.
`scale`	optional numeric specifying the scale parameter of the model, see `scale` in `step`.
`k`	numeric specifying the "weight" of the *equivalent degrees of freedom* (=:`edf`) part in the AIC formula.
`...`	further arguments.

Details

The functions find statistics that resemble `deviance` and `AIC` in constrained ordination. Actually, constrained ordination methods do not have a log-Likelihood, which means that they cannot have AIC and deviance. Therefore you should not use these functions, and if you use them, you should not trust them. If you use these functions, it remains as your responsibility to check the adequacy of the result.

The deviance of `cca` is equal to the Chi-square of the residual data matrix after fitting the constraints. The deviance of `rda` is defined as the residual sum of squares. The deviance function of `rda` is also used for `capscale`. Function `extractAIC` mimics `extractAIC.lm` in translating deviance to AIC.

There is little need to call these functions directly. However, they are called implicitly in `step` function used in automatic selection of constraining variables. You should check the resulting model with some other criteria, because the statistics used here are unfounded. In particular, the penalty `k` is not properly defined, and the default `k = 2` is not justified theoretically. If you have only continuous covariates, the `step` function will base the model building on magnitude of eigenvalues, and the value of `k` only influences the stopping point (but the variables with the highest eigenvalues are not necessarily the most significant in permutation tests in `anova.cca`). If you also have multi-class factors, the value of `k` will have a capricious effect in model building. The `step` function will pass arguments to `add1.cca` and `drop1.cca`, and setting `test = "permutation"` will provide permutation tests of each deletion and addition which can help in judging the validity of the model building.

Value

The `deviance` functions return “deviance”, and `extractAIC` returns effective degrees of freedom and “AIC”.

Note

These functions are unfounded and untested and they should not be used directly or implicitly. Moreover, usual caveats in using `step` are very valid.

Author(s)

Jari Oksanen

References

Godínez-Domínguez, E. & Freire, J. (2003) Information-theoretic approach for selection of spatial and temporal models of community organization. *Marine Ecology Progress Series* **253**, 17–24.

See Also

`cca`, `rda`, `anova.cca`, `step`, `extractAIC`, `add1.cca`, `drop1.cca`.

Examples

```
# The deviance of correspondence analysis equals Chi-square
data(dune)
data(dune.env)
chisq.test(dune)
deviance(cca(dune))
# Backward elimination from a complete model "dune ~ ."
ord <- cca(dune ~ ., dune.env)
ord
step(ord)
# Stepwise selection (forward from an empty model "dune ~ 1")
step(cca(dune ~ 1, dune.env), scope = formula(ord))
# ANOVA: added variable + the first left out
anova(cca(dune ~ Moisture + Management, dune.env), permut=200,
      by = "terms")
```

`distconnected` *Connectedness of Dissimilarities*

Description

Function `distconnected` finds groups that are connected disregarding dissimilarities that are at or above a threshold or `NA`. The function can be used to find groups that can be ordinated together or transformed by `stepacross`. Function `no.shared` returns a logical dissimilarity object, where `TRUE` means that sites have no species in common. This is a minimal structure for `distconnected` or can be used to set missing values to dissimilarities.

Usage

```
distconnected(dis, toolong = 1, trace = TRUE)

no.shared(x)
```

Arguments

`dis`	Dissimilarity data inheriting from class `dist` or a an object, such as a matrix, that can be converted to a dissimilarity matrix. Functions `vegdist` and `dist` are some functions producing suitable dissimilarity data.
`toolong`	Shortest dissimilarity regarded as `NA`. The function uses a fuzz factor, so that dissimilarities close to the limit will be made `NA`, too. If `toolong = 0` (or negative), no dissimilarity is regarded as too long.
`trace`	Summarize results of `distconnected`
`x`	Community data.

Details

Data sets are disconnected if they have sample plots or groups of sample plots which share no species with other sites or groups of sites. Such data sets cannot be sensibly ordinated by any unconstrained method because these subsets cannot be related to each other. For instance, correspondence analysis will polarize these subsets with eigenvalue 1. Neither can such dissimilarities be transformed with `stepacross`, because there is no path between all points, and result will contain `NA`s. Function `distconnected` will find such subsets in dissimilarity matrices. The function will return a grouping vector that can be used for sub-setting the data. If data are connected, the result vector will be all 1s. The connectedness between two points can be defined either by a threshold `toolong` or using input dissimilarities with `NA`s.

Function `no.shared` returns a `dist` structure having value `TRUE` when two sites have nothing in common, and value `FALSE` when they have at least one shared species. This is a minimal structure that can be analysed with `distconnected`. The function can be used to select dissimilarities with no shared species in indices which do not have a fixed upper limit.

Function `distconnected` uses depth-first search (Sedgewick 1990).

Value

Function `distconnected` returns a vector for observations using integers to identify connected groups. If the data are connected, values will be all `1`. Function `no.shared` returns an object of class `dist`.

Author(s)

Jari Oksanen

References

Sedgewick, R. (1990). *Algorithms in C*. Addison Wesley.

See Also

`vegdist` or `dist` for getting dissimilarities, `stepacross` for a case where you may need `distconnected`, and for connecting points `spantree`.

Examples

```
## There are no disconnected data in vegan, and the following uses an
## extremely low threshold limit for connectedness. This is for
## illustration only, and not a recommended practice.
data(dune)
dis <- vegdist(dune)
gr <- distconnected(dis, toolong=0.4)
# Make sites with no shared species as NA in Manhattan dissimilarities
dis <- vegdist(dune, "manhattan")
is.na(dis) <- no.shared(dune)
```

`diversity` *Ecological Diversity Indices and Rarefaction Species Richness*

Description

Shannon, Simpson, and Fisher diversity indices and rarefied species richness for community ecologists.

Usage

```
diversity(x, index = "shannon", MARGIN = 1, base = exp(1))

rarefy(x, sample, se = FALSE, MARGIN = 1)

fisher.alpha(x, MARGIN = 1, se = FALSE, ...)

specnumber(x, MARGIN = 1)
```

Arguments

`x`	Community data, a matrix-like object.
`index`	Diversity index, one of `"shannon"`, `"simpson"` or `"invsimpson"`.
`MARGIN`	Margin for which the index is computed.
`base`	The logarithm `base` used in `shannon`.
`sample`	Subsample size for rarefying community, either a single value or a vector.
`se`	Estimate standard errors.
`...`	Parameters passed to `nlm`

Details

Shannon or Shannon–Weaver (or Shannon–Wiener) index is defined as $H' = -\sum_i p_i \log_b p_i$, where p_i is the proportional abundance of species i and b is the base of the logarithm. It is most popular to use natural logarithms, but some argue for base $b = 2$ (which makes sense, but no real difference).

Both variants of Simpson's index are based on $D = \sum p_i^2$. Choice `simpson` returns $1 - D$ and `invsimpson` returns $1/D$.

Function `rarefy` gives the expected species richness in random subsamples of size `sample` from the community. The size of `sample` should be smaller than total community size, but the function will silently work for larger `sample` as well and return non-rarefied species richness (and standard error = 0). If `sample` is a vector, rarefaction is performed for each sample size separately. Rarefaction can be performed only with genuine counts of individuals. The function `rarefy` is based on Hurlbert's (1971) formulation, and the standard errors on Heck et al. (1975).

`fisher.alpha` estimates the α parameter of Fisher's logarithmic series (see `fisherfit`). The estimation is possible only for genuine counts of individuals. The function can optionally return standard errors of α. These should be regarded only as rough indicators of the accuracy: the confidence limits of α are strongly non-symmetric and the standard errors cannot be used in Normal inference.

Function `specnumber` finds the number of species. With `MARGIN = 2`, it finds frequencies of species. The function is extremely simple, and shortcuts are easy in plain R.

Better stories can be told about Simpson's index than about Shannon's index, and still grander narratives about rarefaction (Hurlbert 1971). However, these indices are all very closely related (Hill 1973), and there is no reason to despise one more than others (but if you are a graduate student, don't drag me in, but obey your Professor's orders). In particular, the exponent of the Shannon index is linearly related to inverse Simpson (Hill 1973) although the former may be more sensitive to rare species. Moreover, inverse Simpson is asymptotically equal to rarefied species richness in sample of two individuals, and Fisher's α is very similar to inverse Simpson.

Value

A vector of diversity indices or rarefied species richness values. With a single `sample` and `se = TRUE`, function `rarefy` returns a 2-row matrix with rarefied richness (`S`) and its standard error (`se`). If `sample` is a vector in `rarefy`, the function returns a matrix with a column for each `sample` size, and if `se = TRUE`, rarefied richness and its standard error are on consecutive lines. With option `se = TRUE`, function `fisher.alpha` returns a data frame with items for α (`alpha`), its approximate standard errors (`se`), residual degrees of freedom (`df.residual`), and the `code` returned by `nlm` on the success of estimation.

Author(s)

Jari Oksanen and Bob O'Hara ⟨bob.ohara@helsinki.fi⟩ (`fisher.alpha`).

References

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 animal population. *Journal of Animal Ecology* **12**, 42–58.

Heck, K.L., van Belle, G. & Simberloff, D. (1975). Explicit calculation of the rarefaction diversity measurement and the determination of sufficient sample size. *Ecology* **56**, 1459–1461.

Hurlbert, S.H. (1971). The nonconcept of species diversity: a critique and alternative parameters. *Ecology* **52**, 577–586.

See Also

Function `renyi` for generalized Rényi diversity and Hill numbers.

Examples

```
data(BCI)
H <- diversity(BCI)
simp <- diversity(BCI, "simpson")
invsimp <- diversity(BCI, "inv")
r.2 <- rarefy(BCI, 2)
alpha <- fisher.alpha(BCI)
pairs(cbind(H, simp, invsimp, r.2, alpha), pch="+", col="blue")
## Species richness (S) and Pielou's evenness (J):
S <- specnumber(BCI) ## rowSums(BCI > 0) does the same...
J <- H/log(S)
```

`dune` *Vegetation and Environment in Dutch Dune Meadows.*

Description

The dune meadow vegetation data, `dune`, has cover class values of 30 species on 20 sites. The corresponding environmental data frame `dune.env` has following entries:

Usage

```
data(dune)
data(dune.env)
```

Format

For `dune`, a data frame of observations of 30 species at 20 sites.

For `dune.env`, a data frame of 20 observations on the following 5 variables:

A1: a numeric vector of thickness of soil A1 horizon.

Moisture: an ordered factor with levels: `1` < `2` < `4` < `5`.

Management: a factor with levels: `BF` (Biological farming), `HF` (Hobby farming), `NM` (Nature Conservation Management), and `SF` (Standard Farming).

Use: an ordered factor of land-use with levels: `Hayfield` < `Haypastu` < `Pasture`.

Manure: an ordered factor with levels: `0` < `1` < `2` < `3` < `4`.

Source

Jongman, R.H.G, ter Braak, C.J.F & van Tongeren, O.F.R. (1987). *Data Analysis in Community and Landscape Ecology*. Pudoc, Wageningen.

Examples

```
data(dune)

data(dune.env)
```

`dune.taxon` *Taxonomic Classification of Dune Meadow Species*

Description

Classification table of the species in the `dune` data set.

Usage

```
data(dune.taxon)
```

Format

A data frame with 30 species (rows) classified into five taxonomic levels (columns).

Details

The classification of vascular plants is adapted from AGP (2003), and that of mosses from Hill et al. (2006).

Note

The data set was made to demonstrate `taxondive`, and will probably be removed after a better example is found.

References

AGP [Angiosperm Phylogeny Group] (2003) An update of the Angiosperm Phylogeny Group classification for the orders and families of flowering plants: AGP II. *Bot. J. Linnean Soc.* **141**: 399–436.

Hill, M.O et al. (2006) An annotated checklist of the mosses of Europe and Macaronesia. *J. Bryology* **28**: 198–267.

Examples

```
data(dune.taxon)
```

`envfit` *Fits an Environmental Vector or Factor onto an Ordination*

Description

The function fits environmental vectors or factors onto an ordination. The projections of points onto vectors have maximum correlation with corresponding environmental variables, and the factors show the averages of factor levels.

Usage

```
## Default S3 method:
envfit(X, P, permutations = 0, strata, choices=c(1,2), ...)
## S3 method for class 'formula':
envfit(formula, data, ...)
## S3 method for class 'envfit':
plot(x, choices = c(1,2), arrow.mul, at = c(0,0), axis = FALSE,
    p.max = NULL, col = "blue", add = TRUE, ...)
## S3 method for class 'envfit':
scores(x, display, choices, ...)
vectorfit(X, P, permutations = 0, strata, choices=c(1,2),
      display = c("sites", "lc"), w = weights(X), ...)
factorfit(X, P, permutations = 0, strata, choices=c(1,2),
      display = c("sites", "lc"), w = weights(X), ...)
```

Arguments

`X`	Ordination configuration.
`P`	Matrix or vector of environmental variable(s).
`permutations`	Number of permutations for assessing significance of vectors or factors.
`formula, data`	Model `formula` and data.
`x`	A result object from `envfit`.
`choices`	Axes to plotted.
`arrow.mul`	Multiplier for vector lengths. The arrows are automatically scaled similarly as in `plot.cca` if this is not given and `add = TRUE`.
`at`	The origin of fitted arrows in the plot. If you plot arrows in other places then origin, you probably have to specify `arrrow.mul`.
`axis`	Plot axis showing the scaling of fitted arrows.
`p.max`	Maximum estimated P value for displayed variables. You must calculate P values with setting `permutations` to use this option.
`col`	Colour in plotting.
`add`	Results added to an existing ordination plot.

`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
`display`	In fitting functions these are ordinary site scores or linear combination scores (`"lc"`) in constrained ordination (`cca`, `rda`, `capscale`). In `scores` function they are either `"vectors"` or `"factors"` (with synonyms `"bp"` or `"cn"`, resp.).
`w`	Weights used in fitting (concerns mainly `cca` and `decorana` results which have nonconstant weights).
`...`	Parameters passed to `scores`.

Details

Function `envfit` finds vectors or factor averages of environmental variables. Function `plot.envfit` adds these in an ordination diagram. If `X` is a `data.frame`, `envfit` uses `factorfit` for `factor` variables and `vectorfit` for other variables. If `X` is a matrix or a vector, `envfit` uses only `vectorfit`. Alternatively, the model can be defined a simplified model `formula`, where the left hand side must be an ordination result object or a matrix of ordination scores, and right hand side lists the environmental variables. The formula interface can be used for easier selection and/or transformation of environmental variables. Only the main effects will be analysed even if interaction terms were defined in the formula.

Functions `vectorfit` and `factorfit` can be called directly. Function `vectorfit` finds directions in the ordination space towards which the environmental vectors change most rapidly and to which they have maximal correlations with the ordination configuration. Function `factorfit` finds averages of ordination scores for factor levels. Function `factorfit` treats ordered and unordered factors similarly.

If `permutations` > 0, the 'significance' of fitted vectors or factors is assessed using permutation of environmental variables. The goodness of fit statistic is squared correlation coefficient (r^2). For factors this is defined as $r^2 = 1 - ss_w/ss_t$, where ss_w and ss_t are within-group and total sums of squares.

User can supply a vector of prior weights `w`. If the ordination object has weights, these will be used. In practise this means that the row totals are used as weights with `cca` or `decorana` results. If you do not like this, but want to give equal weights to all sites, you should set `w = NULL`. The weighted fitting gives similar results to biplot arrows and class centroids in `cca`. For complete similarity between fitted vectors and biplot arrows, you should set `display = "lc"` (and possibly `scaling = 2`).

The results can be accessed with `scores.envfit` function which returns either the fitted vectors scaled by correlation coefficient or the centroids of the fitted environmental variables.

Value

Functions `vectorfit` and `factorfit` return lists of classes `vectorfit` and `factorfit` which have a `print` method. The result object have the following items:

`arrows`	Arrow endpoints from `vectorfit`. The arrows are scaled to unit length.
`centroids`	Class centroids from `factorfit`.
`r`	Goodness of fit statistic: Squared correlation coefficient

`permutations`	Number of permutations.
`pvals`	Empirical P-values for each variable.

Function `envfit` returns a list of class `envfit` with results of `vectorfit` and `envfit` as items.

Function `plot.envfit` scales the vectors by correlation.

Note

Fitted vectors have become the method of choice in displaying environmental variables in ordination. Indeed, they are the optimal way of presenting environmental variables in Constrained Correspondence Analysis `cca`, since there they are the linear constraints. In unconstrained ordination the relation between external variables and ordination configuration may be less linear, and therefore other methods than arrows may be more useful. The simplest is to adjust the plotting symbol sizes (`cex`, `symbols`) by environmental variables. Fancier methods involve smoothing and regression methods that abound in R, and `ordisurf` provides a wrapper for some.

Author(s)

Jari Oksanen. The permutation test derives from the code suggested by Michael Scroggie.

See Also

A better alternative to vectors may be `ordisurf`.

Examples

```
data(varespec)
data(varechem)
library(MASS)
ord <- metaMDS(varespec)
(fit <- envfit(ord, varechem, perm = 1000))
scores(fit, "vectors")
plot(ord)
plot(fit)
plot(fit, p.max = 0.05, col = "red")
## Adding fitted arrows to CCA. We use "lc" scores, and hope
## that arrows are scaled similarly in cca and envfit plots
ord <- cca(varespec ~ Al + P + K, varechem)
plot(ord, type="p")
fit <- envfit(ord, varechem, perm = 1000, display = "lc")
plot(fit, p.max = 0.05, col = "red")
## Class variables, formula interface, and displaying the
## inter-class variability with `ordispider'
data(dune)
data(dune.env)
attach(dune.env)
ord <- cca(dune)
fit <- envfit(ord ~ Moisture + A1, dune.env)
plot(ord, type = "n")
ordispider(ord, Moisture, col="skyblue")
```

```
points(ord, display = "sites", col = as.numeric(Moisture), pch=16)
plot(fit, cex=1.2, axis=TRUE)
```

`fisherfit` *Fit Fisher's Logseries and Preston's Lognormal Model to Abundance Data*

Description

Function `fisherfit` fits Fisher's logseries to abundance data. Function `prestonfit` groups species frequencies into doubling octave classes and fits Preston's lognormal model, and function `prestondistr` fits the truncated lognormal model without pooling the data into octaves.

Usage

```
fisherfit(x, ...)
## S3 method for class 'fisherfit':
confint(object, parm, level = 0.95, ...)
## S3 method for class 'fisherfit':
profile(fitted, alpha = 0.01, maxsteps = 20, del = zmax/5,
    ...)
prestonfit(x, ...)
prestondistr(x, truncate = -1, ...)
## S3 method for class 'prestonfit':
plot(x, xlab = "Frequency", ylab = "Species", bar.col = "skyblue",
    line.col = "red", lwd = 2, ...)
## S3 method for class 'prestonfit':
lines(x, line.col = "red", lwd = 2, ...)
veiledspec(x, ...)
as.fisher(x, ...)
```

Arguments

`x`	Community data vector for fitting functions or their result object for `plot` functions.
`object, fitted`	Fitted model.
`parm`	Not used.
`level`	The confidence level required.
`alpha`	The extend of profiling as significance.
`maxsteps`	Maximum number of steps in profiling.
`del`	Step length.
`truncate`	Truncation point for log-Normal model, in log2 units. Default value -1 corresponds to the left border of zero Octave. The choice strongly influences the fitting results.

`xlab, ylab`	Labels for `x` and `y` axes.
`bar.col`	Colour of data bars.
`line.col`	Colour of fitted line.
`lwd`	Width of fitted line.
`...`	Other parameters passed to functions.

Details

In Fisher's logarithmic series the expected number of species f with n observed individuals is $f_n = \alpha x^n / n$ (Fisher et al. 1943). The estimation follows Kempton & Taylor (1974) and uses function `nlm`. The estimation is possible only for genuine counts of individuals. The parameter α is used as a diversity index, and α and its standard error can be estimated with a separate function `fisher.alpha`. The parameter x is taken as a nuisance parameter which is not estimated separately but taken to be $n/(n + \alpha)$. Helper function `as.fisher` transforms abundance data into Fisher frequency table.

Function `fisherfit` estimates the standard error of α. However, the confidence limits cannot be directly estimated from the standard errors, but you should use function `confint` based on profile likelihood. Function `confint` uses function `confint.glm` of the **MASS** package, using `profile.fisherfit` for the profile likelihood. Function `profile.fisherfit` follows `profile.glm` and finds the τ parameter or signed square root of two times log-Likelihood profile. The profile can be inspected with a `plot` function which shows the τ and a dotted line corresponding to the Normal assumption: if standard errors can be directly used in Normal inference these two lines are similar.

Preston (1948) was not satisfied with Fisher's model which seemed to imply infinite species richness, and postulated that rare species is a diminishing class and most species are in the middle of frequency scale. This was achieved by collapsing higher frequency classes into wider and wider "octaves" of doubling class limits: 1, 2, 3–4, 5–8, 9–16 etc. occurrences. Any logseries data will look like lognormal when plotted this way. The expected frequency f at abundance octave o is defined by $f_o = S_0 \exp(-(\log_2(o) - \mu)^2/2/\sigma^2)$, where μ is the location of the mode and σ the width, both in $\log_2$ scale, and S_0 is the expected number of species at mode. The lognormal model is usually truncated on the left so that some rare species are not observed. Function `prestonfit` fits the truncated lognormal model as a second degree log-polynomial to the octave pooled data using Poisson error. Function `prestondistr` fits left-truncated Normal distribution to $\log_2$ transformed non-pooled observations with direct maximization of log-likelihood. Function `prestondistr` is modelled after function `fitdistr` which can be used for alternative distribution models. The functions have common `print`, `plot` and `lines` methods. The `lines` function adds the fitted curve to the octave range with line segments showing the location of the mode and the width (sd) of the response.

The total extrapolated richness from a fitted Preston model can be found with function `veiledspec`. The function accepts results both from `prestonfit` and from `prestondistr`. If `veiledspec` is called with a species count vector, it will internally use `prestonfit`. Function `specpool` provides alternative ways of estimating the number of unseen species. In fact, Preston's lognormal model seems to be truncated at both ends, and this may be the main reason why its result differ from lognormal models fitted in Rank–Abundance diagrams with functions `rad.lognormal`.

Value

The function `prestonfit` returns an object with fitted `coefficients`, and with observed (`freq`) and fitted (`fitted`) frequencies, and a string describing the fitting `method`. Function `prestondistr` omits the entry `fitted`. The function `fisherfit` returns the result of `nlm`, where item `estimate` is α. The result object is amended with the following items:

`df.residuals`	Residual degrees of freedom.
`nuisance`	Parameter x.
`fisher`	Observed data from `as.fisher`.

Note

It seems that Preston regarded frequencies 1, 2, 4, *etc.*. as "tied" between octaves. This means that only half of the species with frequency 1 were shown in the lowest octave, and the rest were transferred to the second octave. Half of the species from the second octave were transferred to the higher one as well, but this is usually not as large number of species. This practise makes data look more lognormal by reducing the usually high lowest octaves, but is too unfair to be followed. Therefore the octaves used in this function include the upper limit. If you do not accept this, you must change the function yourself. Williamson & Gaston (2005) discuss alternative definitions in detail, and they should be consulted for a critical review of log-Normal model.

Author(s)

Bob O'Hara ⟨bob.ohara@helsinki.fi⟩ (`fisherfit`) and Jari Oksanen.

References

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 animal population. *Journal of Animal Ecology* 12: 42–58.

Kempton, R.A. & Taylor, L.R. (1974). Log-series and log-normal parameters as diversity discriminators for Lepidoptera. *Journal of Animal Ecology* 43: 381–399.

Preston, F.W. (1948) The commonness and rarity of species. *Ecology* 29, 254–283.

Williamson, M. & Gaston, K.J. (2005). The lognormal distribution is not an appropriate null hypothesis for the species–abundance distribution. *Journal of Animal Ecology* 74, 409–422.

See Also

`diversity`, `fisher.alpha`, `radfit`, `specpool`. Function `fitdistr` of **MASS** package was used as the model for `prestondistr`. Function `density` can be used for smoothed "non-parametric" estimation of responses, and `qqplot` is an alternative, traditional and more effective way of studying concordance of observed abundances to any distribution model.

Examples

```
data(BCI)
mod <- fisherfit(BCI[5,])
mod
```

```
plot(profile(mod))
confint(mod)
# prestonfit seems to need large samples
mod.oct <- prestonfit(colSums(BCI))
mod.ll <- prestondistr(colSums(BCI))
mod.oct
mod.ll
plot(mod.oct)
lines(mod.ll, line.col="blue3") # Different
## Smoothed density
den <- density(log2(colSums(BCI)))
lines(den$x, ncol(BCI)*den$y, lwd=2) # Fairly similar to mod.oct
## Extrapolated richness
veiledspec(mod.oct)
veiledspec(mod.ll)
```

`goodness.cca` *Diagnostic Tools for [Constrained] Ordination (CCA, RDA, DCA, CA, PCA)*

Description

Functions `goodness` and `inertcomp` can be used to assess the goodness of fit for individual sites or species. Function `vif.cca` and `alias.cca` can be used to analyse linear dependencies among constraints and conditions. In addition, there are some other diagnostic tools (see 'Details').

Usage

```
## S3 method for class 'cca':
goodness(object, display = c("species", "sites"), choices,
    model = c("CCA", "CA"), statistic = c("explained", "distance"),
    summarize = FALSE, ...)
inertcomp(object, display = c("species", "sites"),
    statistic = c("explained", "distance"), proportional = FALSE)
spenvcor(object)
intersetcor(object)
vif.cca(object)
## S3 method for class 'cca':
alias(object, ...)
```

Arguments

`object`	A result object from `cca`, `rda`, `capscale` or `decorana`.
`display`	Display `"species"` or `"sites"`.
`choices`	Axes shown. Default is to show all axes of the `"model"`.
`model`	Show constrained (`"CCA"`) or unconstrained (`"CA"`) results.

`statistic`	Stastic used: `"explained"` gives the cumulative percentage accounted for, `"distance"` shows the residual distances. Distances are not available for sites in constrained or partial analyses.
`summarize`	Show only the accumulated total.
`proportional`	Give the inertia components as proportional for the corresponding total.
`...`	Other parameters to the functions.

Details

Function `goodness` gives the diagnostic statistics for species or sites. The alternative statistics are the cumulative proportion of inertia accounted for by the axes, and the residual distance left unaccounted for. The conditional ("partialled out") constraints are always regarded as explained and included in the statistics.

Function `inertcomp` decomposes the inertia into partial, constrained and unconstrained components for each site or species. Instead of inertia, the function can give the total dispersion or distances from the centroid for each component.

Function `spenvcor` finds the so-called "species – environment correlation" or (weighted) correlation of weighted average scores and linear combination scores. This is a bad measure of goodness of ordination, because it is sensitive to extreme scores (like correlations are), and very sensitive to overfitting or using too many constraints. Better models often have poorer correlations. Function `ordispider` can show the same graphically.

Function `intersetcor` finds the so-called "interset correlation" or (weighted) correlation of weighted averages scores and constraints. The defined contrasts are used for factor variables. This is a bad measure since it is a correlation. Further, it focuses on correlations between single contrasts and single axes instead of looking at the multivariate relationship. Fitted vectors (`envfit`) provide a better alternative. Biplot scores (see `scores.cca`) are a multivariate alternative for (weighted) correlation between linear combination scores and constraints.

Function `vif.cca` gives the variance inflation factors for each constraint or contrast in factor constraints. In partial ordination, conditioning variables are analysed together with constraints. Variance inflation is a diagnostic tool to identify useless constraints. A common rule is that values over 10 indicate redundant constraints. If later constraints are complete linear combinations of conditions or previous constraints, they will be completely removed from the estimation, and no biplot scores or centroids are calculated for these aliased constraints. A note will be printed with default output if there are aliased constraints. Function `alias` will give the linear coefficients defining the aliased constraints.

Value

The functions return matrices or vectors as is appropriate.

Note

It is a common practise to use `goodness` statistics to remove species from ordination plots, but this may not be a good idea, as the total inertia is not a meaningful concept in `cca`, in particular for rare species.

Function `vif` is defined as generic in package **car** (`vif`), but if you have not loaded that package you must specify the call as `vif.cca`. Variance inflation factor is useful diagnostic tool for detecting nearly collinear constraints, but these are not a problem with algorithm used in this package to fit a constrained ordination.

Author(s)

Jari Oksanen. The `vif.cca` relies heavily on the code by W. N. Venables. `alias.cca` is a simplified version of `alias.lm`.

References

Greenacre, M. J. (1984). Theory and applications of correspondence analysis. Academic Press, London.

Gross, J. (2003). Variance inflation factors. *R News* 3(1), 13–15.

See Also

`cca`, `rda`, `capscale`, `decorana`, `vif`.

Examples

```
data(dune)
data(dune.env)
mod <- cca(dune ~ A1 + Management + Condition(Moisture), data=dune.env)
goodness(mod)
goodness(mod, summ = TRUE)
# Inertia components
inertcomp(mod, prop = TRUE)
inertcomp(mod, stat="d")
# vif.cca
vif.cca(mod)
# Aliased constraints
mod <- cca(dune ~ ., dune.env)
mod
vif.cca(mod)
alias(mod)
with(dune.env, table(Management, Manure))
# The standard correlations (not recommended)
spenvcor(mod)
intersetcor(mod)
```

`goodness.metaMDS` *Goodness of Fit and Shepard Plot for Nonmetric Multidimensional Scaling*

Description

Function `goodness.metaMDS` find goodness of fit measure for points in nonmetric multidimensional scaling, and function `stressplot` makes a `Shepard` diagram.

Usage

```
## S3 method for class 'metaMDS':
goodness(object, dis, ...)
stressplot(object, dis, pch, p.col = "blue", l.col = "red", lwd = 2,
     ...)
```

Arguments

`object`	A result object from `metaMDS` or `isoMDS`.
`dis`	Dissimilarities. Normally this should not used with `metaMDS`, but should be always used with `isoMDS`.
`pch`	Plotting character for points. Default is dependent on the number of points.
`p.col, l.col`	Point and line colours.
`lwd`	Line width.
`...`	Other parameters to functions, e.g. graphical parameters.

Details

Function `goodness.metaMDS` finds a goodness of fit statistic for observations (points). This is defined so that sum of squared values is equal to squared stress. Large values indicate poor fit.

Function `stressplot` is a wrapper to `Shepard` function in **MASS** package. It plots ordination distances against original dissimilarities, and draws a step line of the nonlinear fit. In addition, it adds to the graph two correlation-like statistics on the goodness of fit. The nonmetric fit is based on stress S and defined as $\sqrt{1 - S^2}$. The "linear fit" is the correlation between fitted values and ordination distances.

Both functions can be used both with `metaMDS` and with `isoMDS`. With `metaMDS`, the functions try to reconstruct the dissimilarities using `metaMDSredist`, and dissimilarities should not be given. With `isoMDS` the dissimilarities must be given. In either case, the functions inspect that dissimilarities are consistent with current ordination, and refuse to analyse inconsistent dissimilarities. Function `goodness.metaMDS` is generic in `vegan`, but you must spell its name completely with `isoMDS` which has no class.

Value

Function `goodness` returns a vector of values. Function `stressplot` returns invisibly a `Shepard` object.

Author(s)

Jari Oksanen.

See Also

`metaMDS`, `isoMDS`, `Shepard`.

Examples

```
data(varespec)
mod <- metaMDS(varespec)
stressplot(mod)
gof <- goodness(mod)
gof
plot(mod, display = "sites", type = "n")
points(mod, display = "sites", cex = gof/2)
```

---

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':
summary(object, ...)
## S3 method for class 'humpfit':
predict(object, newdata = NULL, ...)
## 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':
lines(x, segments=101,  ...)
## S3 method for class 'humpfit':
profile(fitted, parm = 1:3, alpha = 0.01, maxsteps = 20, del = zmax/5, ...)
```

Arguments

`mass`	Biomass.
`spno`	Species richness.
`start`	Vector of starting values for all three parameters.
`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.
`x, object, fitted`	Result object of `humpfit`
`newdata`	Values of `mass` used in `predict`. The original data values are used if missing.

`xlab,ylab`	Axis labels in `plot`
`lwd`	Line width
`l.col, p.col`	Line and point colour in `plot`
`type`	Type of `plot`: `"p"` for observed points, `"l"` for fitted lines, `"b"` for both, and `"n"` for only setting axes.
`segments`	Number of segments used for fitted lines.
`parm`	Profiled parameters.
`alpha, maxsteps, del`	Parameters for profiling range and density.
`...`	Other parameters to functions.

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` function) 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. If you have loaded `package` **MASS**, you can use functions `plot.profile.glm`, `pairs.profile.glm` for graphical inspection of the profiles, and `confint.profile.glm` for the profile based confidence intervals.

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.

Author(s)

Jari Oksanen

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

```
##
## Data approximated from Al-Mufti et al. (1977)
##
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))
```

`isomap` *Isometric Feature Mapping Ordination*

Description

The function performs isometric feature mapping which consists of three simple steps: (1) retain only some of the shortest dissimilarities among objects, (2) estimate all dissimilarites as shortest path distances, and (3) perform metric scaling (Tenenbaum et al. 2000).

Usage

```
isomap(dist, ndim=10, ...)
isomapdist(dist, epsilon, k, path = "shortest", fragmentedOK =FALSE, ...)
## S3 method for class 'isomap':
summary(object, axes = 4, ...)
## S3 method for class 'isomap':
plot(x, net = TRUE, n.col = "gray", ...)
rgl.isomap(x, web = "white", ...)
```

Arguments

`dist`	Dissimilariies.
`ndim`	Number of axes in metric scaling (argument `k` in `cmdscale`).
`epsilon`	Shortest dissimilarity retained.
`k`	Number of shortest dissimilariteis retained for a point. If both `epsilon` and `k` are given, `epsilon` will be used.
`path`	Method used in `stepacross` to estimate the shortest path, with alternatives `"shortest"` and `"extended"`.
`fragmentedOK`	What to do if dissimilarity matrix is fragmented. If `TRUE`, analyse the largest connected group, otherwise stop with error.
`x, object`	An `isomap` result object.
`axes`	Number of axes displayed.
`net`	Draw the net of retained dissimilarities.
`n.col`	Colour of drawn net segments.
`web`	Colour of the web in **rgl** graphics.
`...`	Other parameters passed to functions.

Details

The function `isomap` first calls function `isomapdist` for dissimilarity transformation, and then performs metric scaling for the result. All arguments to `isomap` are passed to `isomapdist`. The functions are separate so that the `isompadist` transformation could be easily used with other functions than simple linear mapping of `cmdscale`.

Function `isomapdist` retains either dissimilarities equal or shorter to `epsilon`, or if `epsilon` is not given, at least `k` shortest dissimilarities for a point. Then a complete dissimilarity matrix is reconstructed using `stepacross` using either flexible shortest paths or extended dissimilarities (for details, see `stepacross`).

De'ath (1999) actually published essentially the same method before Tenenbaum et al. (2000), and De'ath's function is available in `xdiss` in package **mvpart**. The differences are that `isomap` introduced the `k` criterion, whereas De'ath only used `epsilon` criterion. In practice, De'ath also retains higher proportion of dissimilarities than typical `isomap`.

In addition to the standard `plot` function, function `rgl.isomap` can make dynamic 3D plots that can be rotated on the screen. The functions is based on `ordirgl`, but it adds the connecting lines. The function passes extra arguments to `scores` and `ordirgl` functions so that you can select axes, or define colours and sizes of points.

Value

Function `isomapdist` returns a dissimilarity object similar to `dist`. Function `isomap` returns an object of class `isomap` with `plot` and `summary` methods. The `plot` function returns invisibly an object of class `ordiplot`. Function `scores` can extract the ordination scores.

Note

Tenenbaum et al. (2000) justify `isomap` as a tool of unfolding a manifold (e.g. a 'Swiss Roll'). Even with a manifold structure, the sampling must be even and dense so that dissimilarities along a manifold are shorter than across the folds. If data do not have such a manifold structure, the results are very sensitive to parameter values.

Author(s)

Jari Oksanen

References

De'ath, G. (1999) Extended dissimilarity: a method of robust estimation of ecological distances from high beta diversity data. *Plant Ecology* 144, 191–199

Tenenbaum, J.B., de Silva, V. & Langford, J.C. (2000) A global network framework for nonlinear dimensionality reduction. *Science* 290, 2319–2323.

See Also

The underlying functions that do the proper work are `stepacross`, `distconnected` and `cmdscale`. Package **mvpart** provides a parallel (but a bit different) implementation (`xdiss`). Moreover, **vegan** function `metaMDS` may trigger `stepacross` transformation, but usually only for longest dissimilarities. The `plot` method of **vegan** minimum spanning tree function (`spantree`) has even more extreme way of isomapping things.

Examples

```
## The following examples also overlay minimum spanning tree to
## the graphics in red.
op <- par(mar=c(4,4,1,1)+0.2, mfrow=c(2,2))
data(BCI)
dis <- vegdist(BCI)
tr <- spantree(dis)
pl <- ordiplot(cmdscale(dis), main="cmdscale")
lines(tr, pl, col="red")
ord <- isomap(dis, k=3)
ord
pl <- plot(ord, main="isomap k=3")
lines(tr, pl, col="red")
pl <- plot(isomap(dis, k=5), main="isomap k=5")
lines(tr, pl, col="red")
pl <- plot(isomap(dis, epsilon=0.45), main="isomap epsilon=0.45")
lines(tr, pl, col="red")
par(op)
## The following command requires user interaction
## Not run:
rgl.isomap(ord, size=4, color="hotpink")
## End(Not run)
```

`linestack` *Plots One-dimensional Diagrams without Overwriting Labels*

Description

Function `linestack` plots vertical one-dimensional plots for numeric vectors. The plots are always labelled, but the labels are moved vertically to avoid overwriting.

Usage

```
linestack(x, labels, cex = 0.8, side = "right", hoff = 2, air = 1.1,
    at = 0, add = FALSE, axis = FALSE, ...)
```

Arguments

`x`	Numeric vector to be plotted.
`labels`	Text labels used instead of default (names of `x`).
`cex`	Size of the labels.
`side`	Put labels to the `"right"` or `"left"` of the axis.
`hoff`	Distance from the vertical axis to the label in units of the width of letter “m”.
`air`	Multiplier to string height to leave empty space between labels.
`at`	Position of plot in horizontal axis.
`add`	Add to an existing plot.
`axis`	Add axis to the plot.
`...`	Other graphical parameters to labels.

Value

The function returns invisibly the shifted positions of labels in user coordinates.

Note

The function always draws labelled diagrams. If you want to have unlabelled diagrams, you can use, e.g., `plot`, `stripchart` or `rug`.

Author(s)

Jari Oksanen

Examples

```
## First DCA axis
data(dune)
ord <- decorana(dune)
linestack(scores(ord, choices=1, display="sp"))
linestack(scores(ord, choices=1, display="si"), side="left", add=TRUE)
title(main="DCA axis 1")
```

`make.cepnames` *Abbreviates a Botanical or Zoological Latin Name into an Eight-character Name*

Description

A standard CEP name has four first letters of the generic name and four first letters of the specific epithet of a Latin name. The last epithet, that may be a subspecific name, is used in the current function. If the name has only one component, it is abbreaviated to eight characters (see `abbreviate`). The returned names are made unique with function `make.unique` which adds numbers to the end of CEP names if needed.

Usage

```
make.cepnames(names)
```

Arguments

`names`	The names to be formatted into CEP names.

Details

Cornell Ecology Programs (CEP) used eight-letter abbreviations for species and site names. In species, the names were formed by taking four first letters of the generic name and four first letters of the specific or subspecific epithet. The CEP names were originally used, because old `FORTRAN IV` did not have `CHARACTER` data type, but text had to be stored in numerical variables, which in popular computers could hold four characters. In modern times, there is no reason for this limitation, but ecologists are used to these names, and they may be practical to avoid congestion in ordination plots.

Value

Function returns CEP names.

Note

The function is simpleminded and rigid. You must write a better one if you need.

Author(s)

Jari Oksanen

See Also

`make.names`, `strsplit`, `substring`, `paste`, `abbreviate`.

Examples

```
make.cepnames(c("Aa maderoi", "Poa sp.", "Cladina rangiferina",
"Cladonia cornuta", "Cladonia cornuta var. groenlandica",
"Cladonia rangiformis", "Bryoerythrophyllum"))
data(BCI)
colnames(BCI) <- make.cepnames(colnames(BCI))
```

`mantel` *Mantel and Partial Mantel Tests for Dissimilarity Matrices*

Description

Function `mantel` finds the Mantel statistic as a matrix correlation between two dissimilarity matrices, and function `mantel.partial` finds the partial Mantel statistic as the partial matrix correlation between three dissimilarity matricies. The significance of the statistic is evaluated by permuting rows and columns of the first dissimilarity matrix.

Usage

```
mantel(xdis, ydis, method="pearson", permutations=1000, strata)
mantel.partial(xdis, ydis, zdis, method = "pearson", permutations = 1000,
    strata)
```

Arguments

`xdis, ydis, zdis`	Dissimilarity matrices or a `dist` objects.
`method`	Correlation method, as accepted by `cor`: `"pearson"`, `"spearman"` or `"kendall"`.
`permutations`	Number of permutations in assessing significance.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.

Details

Mantel statistic is simply a correlation between entries of two dissimilarity matrices (some use cross products, but these are linearly related). However, the significance cannot be directly assessed, because there are $N(N-1)/2$ entries for just N observations. Mantel developed asymptotic test, but here we use permutations of N rows and columns of dissimilarity matrix.

Partial Mantel statistic uses partial correlation conditioned on the third matrix. Only the first matrix is permuted so that the correlation structure between second and first matrices is kept constant. Although `mantel.partial` silently accepts other methods than `"pearson"`, partial correlations will probably be wrong with other methods.

The function uses `cor`, which should accept alternatives `pearson` for product moment correlations and `spearman` or `kendall` for rank correlations.

Value

The function returns a list of class `mantel` with following components:

`Call`	Function call.
`method`	Correlation method used, as returned by `cor.test`.
`statistic`	The Mantel statistic.
`signif`	Empirical significance level from permutations.
`perm`	A vector of permuted values.
`permutations`	Number of permutations.

Note

Legendre & Legendre (1998) say that partial Mantel correlations often are difficult to interpet.

Author(s)

Jari Oksanen

References

The test is due to Mantel, of course, but the current implementation is based on Legendre and Legendre.

Legendre, P. and Legendre, L. (1998) *Numerical Ecology*. 2nd English Edition. Elsevier.

See Also

`cor` for correlation coefficients, `protest` (“Procrustes test”) for an alternative with ordination diagrams, `anosim` and `mrpp` for comparing dissimilarities against classification. For dissimilarity matrices, see `vegdist` or `dist`. See `bioenv` for selecting environmental variables.

Examples

```
## Is vegetation related to environment?
data(varespec)
data(varechem)
veg.dist <- vegdist(varespec) # Bray-Curtis
env.dist <- vegdist(scale(varechem), "euclid")
mantel(veg.dist, env.dist)
mantel(veg.dist, env.dist, method="spear")
```

`metaMDS` *Nonmetric Multidimensional Scaling with Stable Solution from Random Starts, Axis Scaling and Species Scores*

Description

Function `metaMDS` uses `isoMDS` to perform Nonmetric Multidimensional Scaling (NMDS), but tries to find a stable solution using several random starts (function `initMDS`). In addition, it standardizes the scaling in the result, so that the configurations are easier to interpret (function `postMDS`), and adds species scores to the site ordination (function `wascores`).

Usage

```
metaMDS(comm, distance = "bray", k = 2, trymax = 20, autotransform =TRUE,
        noshare = 0.1, wascores = TRUE, expand = TRUE, trace = 1,
        plot = FALSE, previous.best, old.wa = FALSE, ...)
## S3 method for class 'metaMDS':
plot(x, display = c("sites", "species"), choices = c(1, 2),
     type = "p", shrink = FALSE,  ...)
## S3 method for class 'metaMDS':
points(x, display = c("sites", "species"),
        choices = c(1,2), shrink = FALSE, select, ...)
## S3 method for class 'metaMDS':
text(x, display = c("sites", "species"), labels,
        choices = c(1,2), shrink = FALSE, select, ...)
## S3 method for class 'metaMDS':
scores(x, display = c("sites", "species"), shrink = FALSE,
        choices, ...)
metaMDSdist(comm, distance = "bray", autotransform = TRUE, noshare = 0.1,
    trace = 1, commname, zerodist = "fail", distfun = vegdist, ...)
metaMDSiter(dist, k = 2, trymax = 20, trace = 1, plot = FALSE, previous.best,
    ...)
initMDS(x, k=2)
postMDS(X, dist, pc=TRUE, center=TRUE, halfchange=TRUE, threshold=0.8,
        nthreshold=10, plot=FALSE, ...)
metaMDSredist(object, ...)
```

Arguments

`comm`	Community data.
`distance`	Dissimilarity index used in `vegdist`.
`k`	Number of dimensions in `isoMDS`.
`trymax`	Maximum number of random starts in search of stable solution.
`autotransform`	Use simple heuristics for possible data transformation (see below).
`noshare`	Proportion of site pairs with no shared species to trigger `stepacross` to find flexible shortest paths among dissimilarities.
`wascores`	Calculate species scores using function `wascores`.
`expand`	Expand weighted averages of species in `wascores`.
`trace`	Trace the function; `trace = 2` or higher will be more voluminous.
`plot`	Graphical tracing: plot interim results. You may want to set `par(ask = TRUE)` with this option.
`previous.best`	Start searches from a previous solutions. Otherwise use `isoMDS` default for the starting solution.
`old.wa`	Use the old way of calculating WA scores for species: in vegan versions 1.12-5 and 1.11-2 WA scores were based on untransformed data even when data were transformed in analysis, but since then the similar transformation will be used in WA scores as in ordination.
`x`	Dissimilarity matrix for `isoMDS` or `plot` object.
`choices`	Axes shown.
`type`	Plot type: `"p"` for points, `"t"` for text, and `"n"` for axes only.
`display`	Display `"sites"` or `"species"`.
`shrink`	Shrink back species scores if they were expanded originally.
`labels`	Optional test to be used instead of row names.
`select`	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices of displayed items.
`X`	Configuration from multidimensional scaling.
`commname`	The name of `comm`: should not be given if the function is called directly.
`zerodist`	Handling of zero dissimilarities: either `"fail"` or `"add"` a small positive value, or `"ignore"`.
`distfun`	Dissimilarity function. Any function returning a `dist` object and accepting argument `method` can be used (but some extra arguments may cause name conflicts).
`dist`	Dissimilarity matrix used in multidimensional scaling.
`pc`	Rotate to principal components.
`center`	Centre the configuration.
`halfchange`	Scale axes to half-change units.

`threshold`	Largest dissimilarity used in half-change scaling.
`nthreshold`	Minimum number of points in half-change scaling.
`object`	A result object from `metaMDS`.
`...`	Other parameters passed to functions.

Details

Non-metric Multidimensional Scaling (NMDS) is commonly regarded as the most robust unconstrained ordination method in community ecology (Minchin 1987). Functions `initMDS` and `postMDS` together with some other functions are intended to help run NMDS wit `isoMDS` like recommended by Minchin (1987). Function `metaMDS` combines all recommendations into one command for a shotgun style analysis. The steps in `metaMDS` are:

1. Transformation: If the data values are larger than common class scales, the function performs a Wisconsin double standardization using `wisconsin`. If the values look very large, the function also performs `sqrt` transformation. Both of these standardization are generally found to improve the results. However, the limits are completely arbitrary (at present, data maximum 50 triggers `sqrt` and >9 triggers `wisconsin`). If you want to have a full control of the analysis, you should set `autotransform = FALSE` and make explicit standardization in the command.
2. Choice of dissimilarity: For a good result, you should use dissimilarity indices that have a good rank order relation to ordering sites along gradients (Faith et al. 1987). The default is Bray dissimilarity, because it often is the test winner. However, any other dissimilarity index in `vegdist` can be used. Function `rankindex` can be used for finding the test winner for you data and gradients.
3. Step-across dissimilarities: Ordination may be very difficult if a large proportion of sites have no shared species. In this case, the results may be improved with `stepacross` dissimilarities, or flexible shortest paths among all sites. The `stepacross` is triggered by option `noshare`. If you do not like manipulation of original distances, you should set `noshare = 1`.
4. NMDS with random starts: NMDS easily gets trapped into local optima, and you must start NMDS several times from random start to be confident that you have found the global solution. The default in `isoMDS` is to start from metric scaling (with `cmdscale`) which typically is close to a local optimum. The strategy in `metaMDS` is to first run a default `isoMDS`, or use the `previous.best` solution if supplied, and take its solution as the standard (`Run 0`). Then `metaMDS` starts `isoMDS` from several random starts (maximum number is given by `trymax`). If a solution is better (has a lower stress) than the previous standard, it is taken as the new standard. If the solution is better or close to a standard, `metaMDS` compares two solutions using Procrustes analysis using function `procrustes` with option `symmetric = TRUE`. If the two solutions are very similar in their Procrustes `rmse` and the largest residual is very small, the solutions are regarded as convergent and the best one is saved. Please note that the conditions are stringent, and you may have found good and relatively stable solutions although the function is not yet satisfied. Setting `trace = TRUE` will monitor the final stresses, and `plot = TRUE` will display Procrustes overlay plots from each comparison.
5. Scaling of the results: `metaMDS` will run `postMDS` for the final result. Function `postMDS` provides the following ways of “fixing” the indeterminacy of scaling and orientation of axes in NMDS: Centring moves the origin to the average of the axes. Principal components rotate

the configuration so that the variance of points is maximized on first dimension. Half-change scaling scales the configuration so that one unit means halving of community similarity from replicate similarity. Half-change scaling is based on closer dissimilarities where the relation between ordination distance and community dissimilarity is rather linear; the limit is controlled by parameter `threshold`. If there are enough points below this threshold (controlled by the the parameter `nthreshold`), dissimilarities are regressed on distances. The intercept of this regression is taken as the replicate dissimilarity, and half-change is the distance where similarity halves according to linear regression. Obviously the method is applicable only for dissimilarity indices scaled to $0\ldots1$, such as Kulczynski, Bray-Curtis and Canberra indices.

6. Species scores: Function adds the species scores to the final solution as weighted averages using function `wascores` with given value of parameter `expand`. The expansion of weighted averages can be undone with `shrink = TRUE` in `plot` or `scores` functions, and the calculation of species scores can be suppressed with `wascores = FALSE`.

Value

Function `metaMDS` returns an object of class `metaMDS`. The final site ordination is stored in the item `points`, and species ordination in the item `species`. The other items store the information on the steps taken by the function. The object has `print`, `plot`, `points` and `text` methods. Functions `metaMDSdist` and `metaMDSredist` return `vegdist` objects. Function `initMDS` returns a random configuration which is intended to be used within `isoMDS` only. Functions `metaMDSiter` and `postMDS` returns the result of `isoMDS` with updated configuration.

Warning

The calculation of `wascores` for species was changed in **vegan** version 1.12-6. They are now based on the community data transformed similarly as in the ordination. Previously the species scores always were based on the original data. You can re-establish the old behaviour with argument `old.wa = TRUE`.

Note

Function `metaMDS` is a simple wrapper for `isoMDS` and some support functions. You can call these support functions separately for better control of results. Data transformation, dissmilarities and possible `stepacross` are made in function `metaMDSdist` which returns a dissimilarity result. Iterative search (with starting values from `initMDS`) is made in `metaMDSiter`. Processing of result configuration is done in `postMDS`, and species scores added by `wascores`. If you want to be more certain of reaching a global solution, you can compare results from several independent runs. You can also continue analysis from previous results or from your own configuration. Function does not save the used dissimilarity matrix, but `metaMDSredist` tries to reconstruct the used dissimilarities with original data transformation and possible `stepacross`.

Author(s)

Jari Oksanen

References

Faith, D. P, Minchin, P. R. and Belbin, L. (1987). Compositional dissimilarity as a robust measure of ecological distance. *Vegetatio* 69, 57–68.

Minchin, P.R. (1987) An evaluation of relative robustness of techniques for ecological ordinations. *Vegetatio* 71, 145-156.

See Also

`isoMDS`, `decostand`, `wisconsin`, `vegdist`, `rankindex`, `stepacross`, `procrustes`, `wascores`, `ordiplot`.

Examples

```
## The recommended way of running NMDS (Minchin 1987)
##
data(dune)
library(MASS) ## isoMDS
# NMDS
sol <- metaMDS(dune)
sol
plot(sol, type="t")
```

`mite` *Oribatid Mite Data with Explanatory Variables*

Description

Oribatid mite data. 70 soil cores collected by Daniel Borcard in 1989. See Borcard et al. (1992, 1994) for details.

Usage

```
data(mite)
data(mite.env)
data(mite.pcnm)
data(mite.xy)
```

Format

There are three linked data sets: `mite` that contains the data on 35 species of Oribatid mites, `mite.env` that contains environmental data in the same sampling sites, `mite.xy` that contains geographic coordinates, and `mite.pcnm` that contains 22 PCNM base functions (columns) computed from the geographic coordinates of the 70 sampling sites (Borcard & Legendre 2002). The whole sampling area was 2.5 m x 10 m in size.

The fields in the environmental data are:

SubsDens Substrate density (g/L)

WatrCont Water content of the substrate (g/L)

Substrate Substrate type, factor with levels `Sphagn1, Sphagn2 Sphagn3 Sphagn Litter Barepeat Interface`

Shrub Shrub density, an ordered factor with levels `1` < `2` < `3`

Topo Microtopograhy, a factor with levels `Blanket` and `Hummock`

Source

Pierre Legendre

References

Borcard, D., P. Legendre and P. Drapeau. 1992. Partialling out the spatial component of ecological variation. Ecology 73: 1045-1055.

Borcard, D. and P. Legendre. 1994. Environmental control and spatial structure in ecological communities: an example using Oribatid mites (Acari, Oribatei). Environmental and Ecological Statistics 1: 37-61.

Borcard, D. and P. Legendre. 2002. All-scale spatial analysis of ecological data by means of principal coordinates of neighbour matrices. Ecological Modelling 153: 51-68.

Examples

```
data(mite)
```

`mrpp` *Multi Response Permutation Procedure of Within- versus Among-Group Dissimilarities*

Description

Multiple Response Permutation Procedure (MRPP) provides a test of whether there is a significant difference between two or more groups of sampling units.

Usage

```
mrpp(dat, grouping, permutations = 1000, distance = "euclidean",
     weight.type = 1, strata)
```

Arguments

`dat`	data matrix or data frame in which rows are samples and columns are response variable(s), or a dissimilarity object or a symmetric square matrix of dissimilarities.
`grouping`	Factor or numeric index for grouping observations.
`permutations`	Number of permutations to assess the significance of the MRPP statistic, *delta*.
`distance`	Choice of distance metric that measures the dissimilarity between two observations . See `vegdist` for options. This will be used if `dat` was not a dissimilarity structure of a symmetric square matrix.
`weight.type`	choice of group weights. See Details below for options.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.

Details

Multiple Response Permutation Procedure (MRPP) provides a test of whether there is a significant difference between two or more groups of sampling units. This difference may be one of location (differences in mean) or one of spread (differences in within-group distance). Function `mrpp` operates on a `data.frame` matrix where rows are observations and responses data matrix. The response(s) may be uni- or multivariate. The method is philosophically and mathematically allied with analysis of variance, in that it compares dissimilarities within and among groups. If two groups of sampling units are really different (e.g. in their species composition), then average of the within-group compositional dissimilarities ought to be less than the average of the dissimilarities between two random collection of sampling units drawn from the entire population.

The mrpp statistic δ is simply the overall weighted mean of within-group means of the pairwise dissimilarities among sampling units. The correct choice of group weights is currently not clear. The `mrpp` function offers three choices: (1) group size (n), (2) a degrees-of-freedom analogue ($n-1$), and (3) a weight that is the number of unique distances calculated among n sampling units ($n(n-1)/2$).

The `mrpp` algorithm first calculates all pairwise distances in the entire dataset, then calculates δ. It then permutes the sampling units and their associated pairwise distances, and recalculates a δ based on the permuted data. It repeats the permutation step `permutations` times. The significance test is simply the fraction of permuted deltas that are less than the observed delta, with a small sample correction. The function also calculates the change-corrected within-group agreement $A = 1 - \delta/E(\delta)$, where $E(\delta)$ is the expected δ assessed as the average of permutations.

If the first argument `dat` can be interpreted as dissimilarities, they will be used directly. In other cases the function treats `dat` as observations, and uses `vegdist` to find the dissimilarities. The default `distance` is Euclidean as in the traditional use of the method, but other dissimilarities in `vegdist` also are available.

Value

The function returns a list of class mrpp with following items:

`call`	Function call.
`delta`	The overall weighted mean of group mean distances.
`E.delta`	expected delta, under the null hypothesis of no group structure. This is the mean of the permuted deltas.
`Pvalue`	Significance of the test.
`A`	A chance-corrected estimate of the proportion of the distances explained by group identity; a value analogous to a coefficient of determination in a linear model.
`distance`	Choice of distance metric used; the "method" entry of the dist object.
`weight.type`	The choice of group weights used.
`boot.deltas`	The vector of "permuted deltas," the deltas calculated from each of the permuted datasets.
`permutations`	The number of permutations used.

Note

This difference may be one of location (differences in mean) or one of spread (differences in within-group distance). That is, it may find a significant difference between two groups simply because one of those groups has a greater dissimilarities among its sampling units. Most `mrpp` models can be analysed with `adonis` which seems not suffer from the same problems as `mrpp` and is a more robust alternative.

Author(s)

M. Herny H. Stevens ⟨HStevens@muohio.edu⟩ and Jari Oksanen.

References

P. W. Mielke and K. J. Berry. 2001. *Permutation Methods: A Distance Function Approach.* Springer Series in Statistics. Springer.

B. McCune and J. B. Grace. 2002. *Analysis of Ecological Communities.* MjM Software Design, Gleneden Beach, Oregon, USA.

See Also

`anosim` for a similar test based on ranks, and `mantel` for comparing dissimilarities against continuous variables, and `vegdist` for obtaining dissimilarities, `adonis` is a more robust alternative in most cases.

Examples

```
data(dune)
data(dune.env)
dune.mrpp <- mrpp(dune, dune.env$Management)
dune.mrpp

# Save and change plotting parameters
def.par <- par(no.readonly = TRUE)
layout(matrix(1:2,nr=1))

plot(dune.ord <- metaMDS(dune), type="text", display="sites" )
ordihull(dune.ord, dune.env$Management)

with(dune.mrpp, {
  fig.dist <- hist(boot.deltas, xlim=range(c(delta,boot.deltas)),
                 main="Test of Differences Among Groups")
  abline(v=delta);
  text(delta, 2*mean(fig.dist$counts), adj = -0.5,
     expression(bold(delta)), cex=1.5 )  }
)
par(def.par)
```

mso — *Functions for performing and displaying a spatial partitioning of cca or rda results*

Description

The function `mso` adds an attribute `vario` to an object of class `"cca"` that describes the spatial partitioning of the `cca` object and performs an optional permutation test for the spatial independence of residuals. The function `plot.mso` creates a diagnostic plot of the spatial partitioning of the `"cca"` object.

Usage

```
mso(object.cca, object.xy, grain = 1, round.up = FALSE, permutations = FALSE)
msoplot(x, alpha = 0.05, explained = FALSE, ...)
```

Arguments

`object.cca`	An object of class cca, created by the `cca` or `rda` function.
`object.xy`	A vector, matrix or data frame with the spatial coordinates of the data represented by object.cca. Must have the same number of rows as `object.cca$CA$Xbar` (see `cca.object`).
`grain`	Interval size for distance classes.
`round.up`	Determines the choice of breaks. If false, distances are rounded to the nearest multiple of grain. If true, distances are rounded to the upper multiple of grain.
`permutations`	If false, suppresses the permutation test. If an integer, determines the number of permutations for the Mantel test of spatial independence of residual inertia.
`x`	A result object of `mso`.
`alpha`	Significance level for the two-sided permutation test of the Mantel statistic for spatial independence of residual inertia and for the point-wise envelope of the variogram of the total variance. A Bonferroni-type correction can be achieved by dividing the overall significance value (e.g. 0.05) by the number of distance classes.
`explained`	If false, suppresses the plotting of the variogram of explained variance.
`...`	Other arguments passed to functions.

Details

The Mantel test is an adaptation of the function `mantel` of the **vegan** package to the parallel testing of several distance classes. It compares the mean inertia in each distance class to the pooled mean inertia of all other distance classes.

If there are explanatory variables (RDA, CCA, pRDA, pCCA) and a significance test for residual autocorrelation was performed when running the function `mso`, the function `plot.mso` will print an estimate of how much the autocorrelation (based on significant distance classes) causes the global error variance of the regression analysis to be underestimated

Value

The function `mso` returns an amended `cca` or `rda` object with the additional attributes `grain`, `H`, `H.test` and `vario`.

`grain`	The grain attribute defines the interval size of the distance classes .
`H`	H is an object of class 'dist' and contains the geographic distances between observations.
`H.test`	H.test contains a set of dummy variables that describe which pairs of observations (rows = elements of `object$H`) fall in which distance class (columns).
`vario`	The vario attribute is a data frame that contains some or all of the following components for the rda case (cca case in brackets):
H	Distance class as multiples of grain.
Dist	Average distance of pairs of observations in distance class H.
n	Number of unique pairs of observations in distance class H.
All	Empirical (chi-square) variogram of total variance (inertia).
Sum	Sum of empirical (chi-square) variograms of explained and residual variance (inertia).
CA	Empirical (chi-square) variogram of residual variance (inertia).
CCA	Empirical (chi-square) variogram of explained variance (inertia).
pCCA	Empirical (chi-square) variogram of conditioned variance (inertia).
se	Standard error of the empirical (chi-square) variogram of total variance (inertia).
CA.signif	P-value of permutation test for spatial independence of residual variance (inertia).

Note

The function is based on the code published in the Ecological Archives E085-006 (`http://www.esapubs.org/archive/ecol/E085/006/default.htm`).

Author(s)

The responsible author was Helene Wagner.

References

Wagner, H.H. 2004. Direct multi-scale ordination with canonical correspondence analysis. *Ecology* 85: 342–351.

See Also

Function `cca` and `rda`, `cca.object`.

Examples

```
## Reconstruct worked example of Wagner (submitted):
X <- matrix(c(1, 2, 3, 2, 1, 0), 3, 2)
Y <- c(3, -1, -2)
tmat <- c(1:3)
## Canonical correspondence analysis (cca):
Example.cca <- cca(X, Y)
Example.cca <- mso(Example.cca, tmat)
msoplot(Example.cca)
Example.cca$vario

## Correspondence analysis (ca):
Example.ca <- mso(cca(X), tmat)
msoplot(Example.ca)

## Unconstrained ordination with test for autocorrelation
## using oribatid mite data set as in Wagner (2004)
data(mite)
data(mite.env)
data(mite.xy)

mite.cca <- cca(log(mite + 1))
mite.cca <- mso(mite.cca, mite.xy, grain =  1, permutations = 100)
msoplot(mite.cca)
mite.cca

## Constrained ordination with test for residual autocorrelation
## and scale-invariance of species-environment relationships
mite.cca <- cca(log(mite + 1) ~ SubsDens + WatrCont + Substrate + Shrub + Topo, mite.env)
mite.cca <- mso(mite.cca, mite.xy, permutations = 100)
msoplot(mite.cca)
mite.cca
```

`nestedtemp` *Nestedness Indices for Communities of Islands or Patches*

Description

Patches or local communities are regarded as nested if they all could be subsets of the same community. In general, species poor communities should be subsets of species rich communities, and rare species should only occur in species rich communities.

Usage

```
nestedchecker(comm)
nestedn0(comm)
nesteddisc(comm)
nestedtemp(comm, ...)
## S3 method for class 'nestedtemp':
```

```
plot(x, kind = c("temperature", "incidendce"),
    col=rev(heat.colors(100)),  names = FALSE, ...)
```

Arguments

`comm`	Community data.
`x`	Result object for a `plot`.
`col`	Colour scheme for matrix temperatures.
`kind`	The kind of plot produced.
`names`	Label columns and rows in the plot using names in `comm`.
`...`	Other arguments to functions.

Details

The nestedness functions evaluate alternative indices of nestedness. The functions are intended to be used together with Null model communities and used as an argument in `oecosimu` to analyse the nonrdanomness of results.

Function `netstedchecker` gives the number of checkerboard units, or 2x2 submatrices where both species occur once but on different sites (Stone & Roberts 1990). Function `nestedn0` implements nestedness measure N0 which is the number of absences from the sites which are richer than the most pauperate site species occurs (Patterson & Atmar 1986). Function `nesteddisc` implements discrepancy index which is the number of ones that should be shifted to fill a row with ones in a table arranged by species frequencies (Brualdi & Sanderson 1999). Function `nestedtemp` finds the matrix temperature which is defined as the sum of "surprises" in arranged matrix. In arranged unsurprising matrix all species within proportion given by matrix fill are in the upper left corner of the matrix, and the surprise of the absence or presences is the diagonal distance from the fill line (Atmar & Patterson 1993). Function tries to pack species and sites to a low temperature (Rodríguez-Gironés & Santamaria 2006), but this is an iterative procedure, and the temperatures usually vary among runs. Function `nestedtemp` also has a `plot` method which can display either incidences or temperatures of the surprises. Matrix temperature was rather vaguely described (Atmar & Patterson 1993), but Rodríguez-Gironés & Santamaria (2006) are more explicit and their description is used here. However, the results probably differ from other implementations, and users should be cautious in interpreting the results. The details of calculations are explained in the `vignette` *Design decisions and implementation* that you can read using functions `vignette` or `vegandocs`.

Value

The result returned by a nestedness function contains an item called `statistic`, but the other components differ among functions. The functions are constructed so that they can be handled by `oecosimu`.

Author(s)

Jari Oksanen

References

Atmar, W. & Patterson, B.D. (1993). The measurement of order and disorder in the distribution of species in fragmented habitat. *Oecologia* 96, 373–382.

Brualdi, R.A. & Sanderson, J.G. (1999). Nested species subsets, gaps, and discrepancy. *Oecologia* 119, 256–264.

Patterson, B.D. & Atmar, W. (1986). Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biol. J. Linnean Soc.* 28, 65–82.

Rodríguez-Gironés, M.A. & Santamaria, L. (2006). A new algorithm to calculate the nestedness temperature of presence-absence matrices. *J. Biogeogr.* 33, 924–935.

Stone, L. & Roberts, A. (1990). The checkerboard score and species distributions. *Oecologia* 85, 74–79.

Wright, D.H., Patterson, B.D., Mikkelson, G.M., Cutler, A. & Atmar, W. (1998). A comparative analysis of nested subset patterns of species composition. *Oecologia* 113, 1–20.

See Also

In general, the functions should be used with `oecosimu` which generates Null model communities to assess the nonrandomness of nestedness patterns.

Examples

```
data(sipoo)
## Matrix temperature
out <- nestedtemp(sipoo)
out
plot(out)
plot(out, kind="incid")
## Use oecosimu to assess the nonrandomness of checker board units
nestedchecker(sipoo)
oecosimu(sipoo, nestedchecker, "quasiswap")
## Another Null model and standardized checkerboard score
oecosimu(sipoo, nestedchecker, "r00", statistic = "C.score")
```

`oecosimu` *Null Models for Biological Communities*

Description

Null models generate random communities with different criteria to study the significance of nestedness or other community patterns. The function only simulates binary (presence/absence) models with constraint for total number of presences, and optionally for numbers of species and/or species frequencies.

Usage

```
oecosimu(comm, nestfun, method, nsimul = 99, burnin = 0, thin = 1,
   statistic = "statistic",  ...)
commsimulator(x, method, thin=1)
```

Arguments

`comm, x`	Community data.
`nestfun`	Function to analyse nestedness. Some functions are provided in **vegan**, but any function can be used if it accepts the community as the first argument, and returns either a plain number or the result in list item with the name defined in argument `statistic`. See Examples for defining your own functions.
`method`	Null model method. See details.
`nsimul`	Number of simulated null communities.
`burnin`	Number of null communities discarded before proper analysis in sequential methods `"swap"` and `"tswap"`.
`thin`	Number of discarded null communities between two evaluations of nestedness statistic in sequential methods `"swap"` and `"tswap"`.
`statistic`	The name of the statistic returned by `nestedfun`
`...`	Other arguments to functions.

Details

Function `oecosimu` is a wrapper that evaluates a nestedness statistic using function given by `nestfun`, and then simulates a series of null models using `commsimulator`, and evaluates the statistic on these null models. The **vegan** packages contains some nestedness functions that are described separately (`nestedchecker`, `nesteddisc`, `nestedn0`, `nestedtemp`), but many other functions can be used as long as they are meaningful with binary community models. An applicable function must return either the statistic as a plain number, or as a list element `"statistic"` (like `chisq.test`), or in an item whose name is given in the argument `statistic`. The statistic can be a single number (like typical for a nestedness index), or it can be a vector. The vector indices can be used to analyse site (row) or species (column) properties, see `treedive` for an example.

Function `commsimulator` implements null models for community composition. The implemented models are `r00` which maintains the number of presences but fills these anywhere so that neither species (column) nor site (row) totals are preserved. Methods `r0`, `r1` and `r2` maintain the site (row) frequencies. Method `r0` fills presences anywhere on the row with no respect to species (column) frequencies, `r1` uses column marginal frequencies as probabilities, and `r2` uses squared column sums. Methods `r1` and `r2` try to simulate original species frequencies, but they are not strictly constrained. All these methods are reviewed by Wright et al. (1998). Method `c0` maintains species frequencies, but does not honour site (row) frequencies (Jonsson 2001).

The other methods maintain both row and column frequencies. Methods `swap` and `tswap` implement sequential methods, where the matrix is changed only little in one step, but the changed matrix is used as an input if the next step. Methods `swap` and `tswap` inspect random 2x2 submatrices and if they are checkerboard units, the order of columns is swapped. This changes the matrix structure, but does not influence marginal sums (Gotelli & Entsminger 2003). Method `swap` inspects submatrices so long that a swap can be done. Miklós & Podani (2004) suggest that this may lead into biased sequences, since some columns or rows may be more easily swapped, and they suggest trying a fixed number of times and doing zero to many swaps at one step. This method is implemented by method `tswap` or trial swap. Function `commsimulator` makes only one trial swap in time (which probably does nothing), but `oecosimu` estimates how many submatrices are expected before finding a swappable checkerboard, and uses that ratio to thin the results, so that on average one swap will be found per step of `tswap`. However, the checkerboard frequency probably changes during swaps, but this is not taken into account in estimating the `thin`. One swap still changes the matrix only little, and it may be useful to thin the results so that the statistic is only evaluated after `burnin` steps (and `thin`ned).

Methods `quasiswap` and `backtracking` are not sequential, but each call produces a matrix that is independent of previous matrices, and has the same marginal totals as the original data. The recommended method is `quasiswap` which is much faster because it is implemented in C. Method `bactkracking` is provided for comparison, but it is so slow that it may be dropped from future releases of **vegan** (or also implemented in C). Method `quasiswap` (Miklós & Podani 2004) implements a method where matrix is first filled honouring row and column totals, but with integers that may be larger than one. Then the method inspects random 2x2 matrices and performs a quasiswap on them. Quasiswap is similar to ordinary swap, but it also can reduce numbers above one to ones maintaining marginal totals. Method `backtracking` implements a filling method with constraints both for row and column frequencies (Gotelli & Entsminger 2001). The matrix is first filled randomly using row and column frequencies as probabilities. Typically row and column sums are reached before all incidences are filled in. After that begins "backtracking", where some of the points are removed, and then filling is started again, and this backtracking is done so may times that all incidences will be filled into matrix. The `quasiswap` method is not sequential, but it produces a random incidence matrix with given marginal totals.

Value

Function `oecosimu` returns the result of `nestfun` with one added component called `oecosimu`. The `oecosimu` component contains the simulated values of the statistic (item `simulated`), the name of the `method`, two-sided P value and z-value of the statistic based on simulation. The `commsimulator` returns a null model matrix or a swap of the input matrix.

Note

Functions `commsimulator` and `oecosimu` do not have default `nestfun` nor default `method`, because there is no clear natural choice. If you use these methods, you must be able to choose your own strategy. The choice of nestedness index is difficult because the functions seem to imply very different concepts of structure and randomness. The choice of swapping method is also problematic. Method `r00` has some heuristic value of being really random. However, it produces null models which are different from observed communities in most respects, and a "significant" result may simply mean that not all species are equally common (`r0` is similar with this respect). It is also difficult to find justification for `r2`. The methods maintaining both row and column totals only study the community relations, but they can be very slow. Moreover, they regard marginal totals as constraints instead of results of occurrence patterns. You should evaluate timings in small trials (one cycle) before launching an extensive simulation. One swap is fast, but it changes data only little, and you may need long `burnin` and strong `thin`ning in large matrices. You should plot the simulated values to see that they are more or less stationary and there is no trend. Method `quasiswap` is implemented in C and it is much faster than `backtrack`. Method `backtrack` may be removed from later releases of **vegan** because it is slow, but it is still included for comparison.

If you wonder about the name of `oecosimu`, look at journal names in the References (and more in `nestedtemp`).

Author(s)

Jari Oksanen

References

Gotelli, N.J. & Entsminger, N.J. (2001). Swap and fill algorithms in null model analyis: rethinking the knight's tour. *Oecologia* 129, 281–291.

Gotelli, N.J. & Entsminger, N.J. (2003). Swap algorithms in null model analysis. *Ecology* 84, 532–535.

Jonsson, B.G. (2001) A null model for randomization tests of nestedness in species assemblages. *Oecologia* 127, 309–313.

Miklós, I. & Podani, J. (2004). Randomization of presence-absence matrices: comments and new algorithms. *Ecology* 85, 86–92.

Wright, D.H., Patterson, B.D., Mikkelson, G.M., Cutler, A. & Atmar, W. (1998). A comparative analysis of nested subset patterns of species composition. *Oecologia* 113, 1–20.

See Also

`r2dtable` generates table with given marginals but with entries above one. Functions `permatfull` and `permatswap` generate Null models for count data. Function `rndtaxa` (**labdsv** package) randomizes a community table. See also `nestedtemp` (that also discusses other nestedness functions) and `treedive` for another application.

Examples

```
## Use the first eigenvalue of correspondence analysis as an index
## of structure: a model for making your own functions.
data(sipoo)
caeval <- function(x) decorana(x, ira=1)$evals
out <- oecosimu(sipoo, caeval, "swap", burnin=100, thin=10)
out
## Inspect the swap sequence
matplot(t(out$oecosimu$simulated), type="l")
```

ordihull *Add Graphical Items to Ordination Diagrams*

Description

Functions to add convex hulls, arrows, line segments, regular grids of points, 'spider' graphs, ellipses or cluster dendrogram to ordination diagrams. The ordination diagrams can be produced by `vegan` `plot.cca`, `plot.decorana` or `ordiplot`.

Usage

```
ordihull(ord, groups, display = "sites", draw = c("lines","polygon"),
         show.groups, ...)
ordiarrows(ord, groups, levels, replicates, display = "sites",
         show.groups, startmark, ...)
ordisegments(ord, groups, levels, replicates, display = "sites",
         show.groups, ...)
ordigrid(ord, levels, replicates, display = "sites",  ...)
ordispider(ord, groups, display="sites", w = weights(ord, display),
         show.groups, ...)
ordiellipse(ord, groups, display="sites", kind = c("sd","se"), conf,
         draw = c("lines","polygon"), w = weights(ord, display),
         show.groups, ...)
ordicluster(ord, cluster, prune = 0, display = "sites",
         w = weights(ord, display), ...)
```

Arguments

`ord`	An ordination object or an `ordiplot` object.
`groups`	Factor giving the groups for which the graphical item is drawn.
`levels, replicates`	Alternatively, regular groups can be defined with arguments `levels` and `replicates`, where `levels` gives the number of groups, and `replicates` the number of successive items at the same group.
`display`	Item to displayed.
`draw`	Use either `lines` or `polygon` to draw the line. Graphical parameters are passed to both. The main difference is that `polygons` may be filled and non-transparent.
`show.groups`	Show only given groups. This can be a vector, or `TRUE` if you want to show items for which condition is `TRUE`. This argument makes it possible to use different colours and line types for groups. The default is to show all groups.
`startmark`	plotting characer used to mark the first item. The default is to use no mark, and for instance, `startmark = 1` will draw a circle. For other plotting characters, see `pch` in `points`.

`w`	Weights used to find the average within group. Weights are used automatically for `cca` and `decorana` results, unless undone by the user. `w=NULL` sets equal weights to all points.
`kind`	Whether standard deviations of points (`sd`) or standard deviations of their (weighted) averages (`se`) are used.
`conf`	Confidence limit for ellipses, e.g. 0.95. If given, the corresponding `sd` or `se` is multiplied with the corresponding value found from the Chi-squared distribution with 2df.
`cluster`	Result of hierarchic cluster analysis, such as `hclust` or `agnes`.
`prune`	Number of upper level hierarchies removed from the dendrogram. If `prune` > 0, dendrogram will be disconnected.
`...`	Parameters passed to graphical functions such as `lines`, `segments`, `arrows`, `polygon` or to `scores` to select axes and scaling etc.

Details

Function `ordihull` draws `lines` or `polygon`s for the convex hulls found by function `chull` encircling the items in the groups.

Function `ordiarrows` draws `arrows` and `ordisegments` draws line `segments` between successive items in the groups. Function `ordigrid` draws line `segments` both within the groups and for the corresponding items among the groups.

Function `ordispider` draws a 'spider' diagram where each point is connected to the group centroid with `segments`. Weighted centroids are used in the correspondence analysis methods `cca` and `decorana` or if the user gives the weights in the call. If `ordispider` is called with `cca` or `rda` result without `groups` argument, the function connects each 'WA' scores to the correspoding 'LC' score.

Function `ordiellipse` draws `lines` or `polygon`s for dispersion `ellipse` using either standard deviation of point scores or standard error of the (weighted) average of scores, and the (weighted) correlation defines the direction of the principal axis of the ellipse. The function requires package **ellipse**. An ellipsoid hull can be drawn with function `ellipsoidhull` of package **cluster**.

Function `ordicluster` overlays a cluster dendrogram onto ordination. It needs the result from a hierarchic clustering such as `hclust` or `agnes`, or other with a similar structure. Function `ordicluster` connects cluster centroids to each other with line `segments`. Function uses centroids of all points in the clusters, and is therefore similar to average linkage methods.

Note

These functions add graphical items to ordination graph: You must draw a graph first.

Author(s)

Jari Oksanen

See Also

The functions pass parameters to basic graphical functions, and you may wish to change the default values in `arrows`, `lines`, `segments` and `polygon`. You can pass parameters to `scores` as well. Other underlying functions are `chull` and `ellipse`.

Examples

```
data(dune)
data(dune.env)
mod <- cca(dune ~ Moisture, dune.env)
attach(dune.env)
## pass non-graphical arguments without warnings
plot(mod, type="n", scaling = 3)
ordihull(mod, Moisture, scaling = 3)
ordispider(mod, col="red", scaling = 3)
plot(mod, type = "p", display="sites")
ordicluster(mod, hclust(vegdist(dune)), prune=3, col = "blue")
# The following is not executed automatically because it needs
# a non-standard library `ellipse'.
## Not run:
ordiellipse(mod, Moisture, kind="se", conf=0.95, lwd=2, col="blue")
## End(Not run)
```

`ordilabel` *Add Text on Non-transparent Label to an Ordination Plot*

Description

Function `ordilabel` is similar to `text`, but the text is on an opaque label. This can help in crowded ordination plots: you still cannot see all text labels, but at least the uppermost are readable. Argument `priority` helps to make the most important labels visible.

Usage

```
ordilabel(x, display, labels, choices = c(1, 2), priority, cex = 0.8,
    fill = "white", border = NULL, ...)
```

Arguments

`x`	An ordination object an any object known to `scores`.
`display`	Kind of scores displayed (passed to `scores`).
`labels`	Optional text used in plots. If this is not given, the text is found from the ordination object.
`choices`	Axes shown (passed to `scores`).
`priority`	Vector of the same length as the number of labels. The items with high priority will be plotted uppermost.
`cex`	Character expansion for the text (passed to `text`).
`fill`	Background colour of the labels (the `col` argument of `polygon`).
`border`	The colour and visibilit of the border of the label as defined in `polygon`).
`...`	Other arguments (passed to `text`).

Details

The function may be useful with crowded ordination plots, in particular together with argument `priority`. You will not see all text labels, but at least some are readable. Other alternatives to crowded plots are `identify.ordiplot`, `orditorp` and `orditkplot`.

Author(s)

Jari Oksanen

See Also

`scores`, `polygon`, `text`. The function is modelled after `s.label` in **ade4** package.

Examples

```
data(dune)
ord <- cca(dune)
plot(ord, type = "n")
ordilabel(ord, dis="sites", cex=1.2, font=3, fill="hotpink", col="blue")
## You may prefer separate plots, but here species as well
ordilabel(ord, dis="sp", font=2, priority=colSums(dune))
```

`ordiplot` *Alternative plot and identify Functions for Ordination*

Description

Ordination plot function especially for congested plots. Function `ordiplot` always plots only unlabelled points, but `identify.ordiplot` can be used to add labels to selected site, species or constraint points. Function `identify.ordiplot` can be used to identify points from `plot.cca`, `plot.decorana`, `plot.procrustes` and `plot.rad` as well.

Usage

```
ordiplot(ord, choices = c(1, 2), type="points", display, xlim, ylim, ...)
## S3 method for class 'ordiplot':
identify(x, what, labels,  ...)
## S3 method for class 'ordiplot':
points(x, what, select, ...)
## S3 method for class 'ordiplot':
text(x, what, labels, select, ...)
```

Arguments

`ord`	A result from an ordination.
`choices`	Axes shown.
`type`	The type of graph which may be `"points"`, `"text"` or `"none"` for any ordination method.
`display`	Display only "sites" or "species". The default for most methods is to display both, but for `cca`, `rda` and `capscale` it is the same as in `plot.cca`.
`xlim, ylim`	the x and y limits (min,max) of the plot.
`...`	Other graphical parameters.
`x`	A result object from `ordiplot`.
`what`	Items identified in the ordination plot. The types depend on the kind of plot used. Most methods know `sites` and `species`, functions `cca` and `rda` know in addition `constraints` (for 'LC' scores), `centroids` and `biplot`, and `plot.procrustes` ordination plot has `heads` and `points`.
`labels`	Optional text used for labels. Row names will be used if this is missing.
`select`	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices of displayed items.

Details

Function `ordiplot` draws an ordination diagram using black circles for sites and red crosses for species. It returns invisibly an object of class `ordiplot` which can be used by `identify.ordiplot` to label selected sites or species, or constraints in `cca` and `rda`.

The function can handle output from several alternative ordination methods. For `cca`, `rda` and `decorana` it uses their `plot` method with option `type = "points"`. In addition, the `plot` functions of these methods return invisibly an `ordiplot` object which can be used by `identify.ordiplot` to label points. For other ordinations it relies on `scores` to extract the scores.

For full user control of plots, it is best to call `ordiplot` with `type = "none"` and save the result, and then add sites and species using `points.ordiplot` or `text.ordiplot` which both pass all their arguments to the corresponding default graphical functions.

Value

Function `ordiplot` returns invisibly an object of class `ordiplot` with items `sites`, `species` and `constraints` (if these are available in the ordination object). Function `identify.ordiplot` uses this object to label the point.

Note

The purpose of these functions is to provide similar functionality as the `plot`, `plotid` and `specid` methods in library `labdsv`. The functions are somewhat limited in parametrization, but you can call directly the standard `identify` and `plot` functions for a better user control.

Author(s)

Jari Oksanen

See Also

identify for basic operations, plot.cca, plot.decorana, plot.procrustes which also produce objects for identify.ordiplot and scores for extracting scores from non-vegan ordinations.

Examples

```
# Draw a cute NMDS plot from a non-vegan ordination (isoMDS).
# Function metaMDS would be an easier alternative.
data(dune)
dune.dis <- vegdist(wisconsin(dune))
library(MASS)
dune.mds <- isoMDS(dune.dis)
dune.mds <- postMDS(dune.mds, dune.dis)
dune.mds$species <- wascores(dune.mds$points, dune, expand = TRUE)
fig <- ordiplot(dune.mds, type = "none")
points(fig, "sites", pch=21, col="red", bg="yellow")
text(fig, "species", col="blue", cex=0.9)
# Default plot of the previous using identify to label selected points
## Not run:
fig <- ordiplot(dune.mds)
identify(fig, "spec")
## End(Not run)
```

---

ordiplot3d *Three-Dimensional and Dynamic Ordination Graphics*

Description

Function ordiplot3d displays three-dimensional ordination graphics using scatterplot3d. Function ordirgl displays three-dimensional dynamic ordination graphs which can be rotated and zoomed into using rgl package. Both work with all ordination results form vegan and all ordination results known by scores function.

Usage

```
ordiplot3d(object, display = "sites", choices = 1:3, ax.col = 2,
        arr.len = 0.1, arr.col = 4, envfit, xlab, ylab, zlab, ...)
ordirgl(object, display = "sites", choices = 1:3, type = "p",
        ax.col = "red", arr.col = "yellow", text, envfit, ...)
orglpoints(object, display = "sites", choices = 1:3, ...)
orgltext(object, text, display = "sites", choices = 1:3, justify = "center",
        adj = 0.5, ...)
orglsegments(object, groups, display = "sites", choices = 1:3, ...)
orglspider(object, groups, display = "sites", w = weights(object, display),
        choices = 1:3, ...)
```

Arguments

`object`	An ordination result or any object known by `scores`.
`display`	Display `"sites"` or `"species"` or other ordination object recognized by `scores`.
`choices`	Selected three axes.
`arr.len`	'Length' (width) of arrow head passed to `arrows` function.
`arr.col`	Colour of biplot `arrows` and centroids of environmental variables.
`type`	The type of plots: `"p"` for points or `"t"` for text labels.
`ax.col`	Axis colour (concerns only the crossed axes through the origin).
`text`	Text to override the default with `type = "t"`.
`envfit`	Fitted environmental variables from `envfit` displayed in the graph.
`xlab, ylab, zlab`	Axis labels passed to `scatterplot3d`. If missing, labels are taken from the ordination result. Set to `NA` to supress labels.
`justify, adj`	Text justification passed to `rgl.texts`. One of these is used depending on the versionof **rgl** installed.
`groups`	Factor giving the groups for which the graphical item is drawn.
`w`	Weights used to find the average within group. Weights are used automatically for `cca` and `decorana` results, unless undone by the user. `w=NULL` sets equal weights to all points.
`...`	Other parameters passed to graphical functions.

Details

Both function display three-dimensional ordination graphics. Function `ordiplot3d` plots static scatter diagrams using `scatterplot3d`. Function `ordirgl` plots dynamic graphics using OpenGL in `rgl`. Both functions use most default settings of underlying graphical functions, and you must consult their help pages to change graphics to suit your taste (see `scatterplot3d`, `rgl`, `rgl.points`,`rgl.texts`). Both functions will display only one selected set of `scores`, typically either `"sites"` or `"species"`, but for instance `cca` also has `"lc"` scores. In constrained ordination (`cca`, `rda`, `capscale`), biplot arrows and centroids are always displayed similarly as in two-dimensional plotting function `plot.cca`. Alternatively, it is possible to display fitted environmental vectors or class centroids from `envfit` in both graphs. These are displayed similarly as the results of constrained ordination, and they can be shown only for non-constrained ordination. The user must remember to specify at least three axes in `envfit` if the results are used with these functions.

Function `ordiplot3d` plots only points. However, it returns invisibly an object inheriting from `ordiplot` so that you can use `identify.ordiplot` to identify `"points"` or `"arrows"`. The underlying `scatterplot3d` function accepts `type = "n"` so that only the axes, biplot arrows and centroids of environmental variables will be plotted, and the ordination scores can be added with `text.ordiplot` or `points.ordiplot`. Further, you can use any functions from the `ordihull` family with the invisble result of `ordiplot3d`, but you must remember to specify the `display` as `"points"` or `"arrows"`. To change the viewing angle, orientation etc. you must see `scatterplot3d`.

Function `ordigl` makes a dynamic three-dimensional graph that can be rotated with mouse, and zoomed into with mouse buttons or wheel (but Mac users with one-button mouse should see `rgl.viewpoint`), or try ctrl-button. MacOS X users must start `X11` before calling `rgl` commands. Function `ordirgl` uses default settings, and you should consult the underlying functions `rgl.points`, `rgl.texts` to see how to control the graphics. Function `ordirgl` always cleans its graphic window before drawing. Functions `orglpoints` adds points and `orgltext` adds text to existing `ordirgl` windows. In addition, function `orglsegments` combines points within `"groups"` with line segments similarly as `ordisegments`. Function `orglspider` works similarly as `ordispider`: it connects points to their weighted centroid within `"groups"`, and in constrained ordination it can connect `"wa"` or weighted averages scores to corresponding `"lc"` or linear combination scores if `"groups"` is missing. In addition, basic `rgl` functions `rgl.points`, `rgl.texts`, `rgl.lines` and many others can be used.

Value

Function `ordiplot3d` returns invisibly an object of class `"ordiplot3d"` inheriting from `ordiplot`. The return object will contain the coordinates projected onto two dimensions for `"points"`, and possibly for the heads of `"arrows"` and `"centroids"` of environmental variables. Functions like `identify.ordiplot`, `points.ordiplot`, `text.ordiplot` can use this result, as well as `ordihull` and other functions documented with the latter. In addition, the result will contain the object returned by `scatterplot3d`, including function `xyz.converter` which projects three-dimensional coordinates onto the plane used in the current plot. Function `ordirgl` returns nothing.

Warning

Function `ordirgl` uses OpenGL package `rgl` which may not be functional in all platforms, and can crash R in some: use `save.image` before trying `ordirgl`. Mac users must start `X11` (and first install `X11` and some other libraries) before being able to use `rgl`. It seems that `rgl.texts` does not always position the text like supposed, and it may be safe to verify text location with corresponding points.

Note

The user interface of **rgl** changed in version 0.65, but the `ordirgl` functions do not yet fully use the new capablities. However, they should work both in old and new versions of **rgl**.

Author(s)

Jari Oksanen

See Also

`scatterplot3d`, `rgl`, `rgl.points`, `rgl.texts`, `rgl.viewpoint`, `ordiplot`, `identify.ordiplot`, `text.ordiplot`, `points.ordiplot`, `ordihull`, `plot.cca`, `envfit`.

Examples

```
## Examples are not run, because they need non-standard packages
## 'scatterplot3d' and 'rgl' (and the latter needs user interaction).
```

```
#####
#### Default 'ordiplot3d'
## Not run:
data(dune)
data(dune.env)
ord <- cca(dune ~ A1 + Moisture, dune.env)
ordiplot3d(ord)
#### A boxed 'pin' version
ordiplot3d(ord, type = "h")
#### More user control
pl <- ordiplot3d(ord, angle=15, type="n")
points(pl, "points", pch=16, col="red", cex = 0.7)
#### identify(pl, "arrows", col="blue") would put labels in better positions
text(pl, "arrows", col="blue", pos=3)
text(pl, "centroids", col="blue", pos=1, cex = 1.2)
#### ordirgl
ordirgl(ord, size=2)
ordirgl(ord, display = "species", type = "t")
rgl.quit()
## End(Not run)
```

`ordipointlabel` *Ordination Plots with Points and Optimized Locations for Text*

Description

The function `ordipointlabel` produces ordination plots with points and text label to the points. The points are in the exact location given by the ordination, but the function tries to optimize the location of the text labels to minimize overplotting text. The function may be useful with moderatly crowded ordination plots.

Usage

```
ordipointlabel(x, display = c("sites", "species"), choices = c(1, 2),
   col = c(1, 2),  pch = c("o", "+"), font = c(1, 1),
   cex = c(0.8, 0.8), add = FALSE, ...)
```

Arguments

`x`	A result object from ordination.
`display`	Scores displayed in the plot.
`choices`	Axes shown.
`col, pch, font, cex`	Colours, point types, font style and character expansion for each kind of scores displayed in the plot. These should be vectors of the same length as the number of items in `display`.
`add`	Add to an existing plot.
`...`	Other rguments passed to `points` and `text`.

Details

The function uses simulated annealing (`optim`, `method = "SANN"`) to optimize the location of the text labels to the points. There are eight possible locations: up, down, sides and corners. There is a weak preference to text right above the point, and a weak avoidance of corner positions. The exact locations and the goodness of solution varies between runs, and there is no guarantee of finding the global optimum. The optimization can take a long time in difficult cases with a high number of potential overlaps. Several sets of scores can be displayed in one plot.

The function is modelled after `pointLabel` in **maptools** package (which has chained dependencies of `S4` packages).

Value

The function returns invisibly an object of class `ordipointlabel` with items `xy` for coordinates of points, `labels` for coordinates of labels, items `pch`, `cex` and `font` for graphical parameters of each point or label. In addition, it returns the result of `optim` as an attribute `"optim"`. The unit of overlap is the area of character `"m"`, and with variable `cex` it is the smallest alternative. The result object inherits from `orditkplot` result, and can be replotted with its `plot` command. It may be possible to further edit the result object with `orditkplot`, but for good results it is necessary that the points span the whole horizontal axis without empty margins.

Note

The function is designed for ordination graphics, and the optimization works properly with plots of isometric aspect ratio.

Author(s)

Jari Oksanen

References

See `pointLabel` for references.

See Also

`pointLabel` for the model implementation, and `optim` for the optimization.

Examples

```
data(varespec)
ord <- cca(varespec)
ordipointlabel(ord)
```

`ordiresids` *Plots of Residuals and Fitted Values for Constrained Ordination*

Description

The function provides `plot.lm` style diagnostic plots for the results of constrained ordination from `cca`, `rda` and `capscale`. Normally you do not need these plots, because ordination is descriptive and does not make assumptions on the distribution of the residuals. However, if you permute residuals in significance tests (`anova.cca`), you may be interested in inspecting that the residuals really are exchangeable and independent of fitted values.

Usage

```
ordiresids(x, kind = c("residuals", "scale", "qqmath"),
   residuals = "working", type = c("p", "smooth", "g"),
   formula, ...)
```

Arguments

`x`	Ordination result from `cca`, `rda` or `capscale`.
`kind`	The type of plot: `"residuals"` plot residuals against fitted values, `"scale"` the square root of absolute residuals against fitted values, and `"qqmath"` the residuals against expected distribution (defaults `qnorm`), unless defined differently in the `formula` argument).
`residuals`	The kind of residuals and fitted values. The argument is passed on to `fitted.cca` with alternatives `"working"` and `"response"`.
`type`	The type of plot. The argument is passed on to **lattice** functions.
`formula`	Formula to override the default plot. The formula can contain items `Fitted`, `Residuals`, `Species` and `Sites` (provided that names of species and sites are available in the ordination result).
`...`	Other arguments passed to **lattice** functions.

Details

The default plots are similar as in `plot.lm`, but they use `Lattice` functions `xyplot` and `qqmath`. The alternatives have default formulas but these can be replaced by the user. The elements available in formula or in the `groups` argument are `Fitted`, `Residuals`, `Species` and `Sites`.

Value

The function return a `Lattice` object that can displayed as plot.

Author(s)

Jari Oksanen

See Also

`plot.lm`, `Lattice`, `xyplot`, `qqmath`.

Examples

```
data(varespec)
data(varechem)
mod <- cca(varespec ~ Al + P + K, varechem)
ordiresids(mod)
```

`ordisurf` *Fit and Plot Smooth Surfaces of Variables on Ordination.*

Description

Function `ordisurf` fits a smooth surface for given variable and plots the result on ordination diagram.

Usage

```
ordisurf(x, y, choices=c(1, 2), knots=10, family="gaussian", col="red",
     thinplate = TRUE, add = FALSE, display = "sites",
     w = weights(x), main, nlevels = 10, levels, labcex = 0.6,  ...)
```

Arguments

`x`	Ordination configuration, either a matrix or a result known by `scores`.
`y`	Variable to be plotted.
`choices`	Ordination axes.
`knots`	Number of initial knots in `gam` (one more than degrees of freedom).
`family`	Error distribution in `gam`.
`col`	Colour of contours.
`thinplate`	Use thinplate splines in `gam`.
`add`	Add contours on an existing diagram or draw a new plot.
`display`	Type of scores known by `scores`: typically "sites" for ordinary site scores or "lc" for linear combination scores.
`w`	Prior weights on the data. Concerns mainly `cca` and `decorana` results which have nonconstant weights.
`main`	The main title for the plot, or as default the name of plotted variable in a new plot.
`nlevels, levels`	Either a vector of `levels` for which contours are drawn, or suggested number of contours in `nlevels` if `levels` are not supplied.
`labcex`	Label size in contours. Setting this zero will suppress labels.
`...`	Other graphical parameters.

Details

Function `ordisurf` fits a smooth surface using thinplate splines in `gam`, and uses `predict.gam` to find fitted values in a regular grid. Function plots the fitted contours with convex hull of data poitns either over an existing ordination diagram or draws a new plot The function uses `scores` to extract ordination scores, and `x` can be any result object known by that function.

User can supply a vector of prior weights `w`. If the ordination object has weights, these will be used. In practise this means that the row totals are used as weights with `cca` or `decorana` results. If you do not like this, but want to give equal weights to all sites, you should set `w = NULL`. The behaviour is consistent with `envfit`. For complete accordance with constrained `cca`, you should set `display = "lc"` (and possibly `scaling = 2`).

Value

Function is usually called for its side effect of drawing the contour plot, but it returns the result object of `gam`.

Note

The default is to use thinplate splines. These make sense in ordination as they have equal smoothing in all directions and are rotation invariant.

Author(s)

Dave Roberts and Jari Oksanen

See Also

For basic routines `gam`, and `scores`. Function `envfit` provides a more traditional and compact alternative.

Examples

```
data(varespec)
data(varechem)
library(MASS)
vare.dist <- vegdist(varespec)
vare.mds <- isoMDS(vare.dist)
with(varechem, ordisurf(vare.mds, Baresoil))
## Cover of Cladina arbuscula
with(varespec, ordisurf(vare.mds, Cla.arb, family=quasipoisson))
```

`orditkplot` *Ordination Plot with Movable Labels*

Description

Function `orditkplot` produces an editable ordination plot with points and labels. The labels can be moved with mouse, and the edited plot can be saved as an encapsulated postscript file or exported via R `plot` function to other graphical formats, or saved in the R session for further processing.

Usage

```
orditkplot(x, display = "species", choices = 1:2, width, xlim, ylim,
   tcex = 0.8, tcol, pch = 1,  pcol, pbg, pcex = 0.7, labels,  ...)
## S3 method for class 'orditkplot':
plot(x, ...)
## S3 method for class 'orditkplot':
points(x, ...)
## S3 method for class 'orditkplot':
text(x, ...)
## S3 method for class 'orditkplot':
scores(x, display, ...)
```

Arguments

`x`	An ordination result or any other object that `scores` can handle, or for the `plot` function the object dumped from the interactive `orditkplot` session.
`display`	Type of `scores` displayed. For ordination scores this typically is either `"species"` or `"sites"`, and for `orditkplot` result it is either `"points"` or `"labels"`.
`choices`	Axes displayed.
`width`	Width of the plot in inches; defaults to the current width of the graphical device.
`xlim, ylim`	x and y limits for plots: points outside these limits will be completely removed.
`tcex`	Character expansion for text labels.
`tcol`	Colour of text labels.
`pch, pcol, pbg`	Point type and outline and fill colours. Defaults `pcol="black"` and `pbg="transparent"`. Argument `pbg` has an effect only in filled plotting characters `pch = 21` to `25`.
`pcex`	Expansion factor for point size.
`labels`	Labels used instead of row names.
`...`	Other arguments passed to the function. These can be graphical parameters (see `par`) used in the plot, or extra arguments to `scores`. These arguments are ignored in `plot`, but honoured in `text` and `points`.

Details

Function `orditkplot` uses **tcltk** package to draw Tcl/Tk based ordination graphics with points and labels. The function opens an editable canvas with fixed points, but the labels can be dragged with mouse to better positions or edited. In addition, it is possible to zoom to a part of the graph.

The function knows the following mouse operations:

- **Left mouse button** can be used to move labels to better positions. A line will connect a label to the corresponding point.
- **Double clicking left mouse button** opens a window where the label can be edited. After editing the label, hit the Return key.
- **Right mouse button** (or alternatively, Shift-Mouse button with one-button mouse) can be used for zooming to a part of the graph. Keeping the mouse button down and dragging will draw a box of the zoomed area, and after releasing the button, a new plot window will be created (this is still preliminary: all arguments are not passed to the new plot).

In addition there are buttons for the following tasks: **Copy to EPS** copies the current plot to an encapsulated postscript (eps) file using standard Tcl/Tk utilities. The faithfullness of this copy is system dependent. Button **Export plot** uses `plot.orditkplot` function to redraw the plot into graphical file formats. Depending on the system, the following graphical formats may be available: eps, pdf, png, jpeg or bmp. The file type is deduced from the file suffix or the selection of the file type in the dialog box. Alternatively, the same dialog can be used to save the plot to an editable `xfig` file. Button **Dump to R** writes the edited coordinates of labels and points to the R session for further processing, and the `plot.orditkplot` function can be used to display the results. For faithful replication of the plot, the graph must have similar dimensions as the `orditkplot` canvas had originally. The `plot` function cannot be configured, but it uses the same settings as the original Tcl/Tk plot. However, `points` and `text` functions are fully configurable, and unaware of the original Tcl/Tk plot settings (probably you must set `cex` at least to get a decent plot). Finally, button **Dismiss** closes the window.

The produced plot will have equal aspect ratio. The width of the horizontal axis is fixed, but vertical axes will be scaled to needed height, and you can use scrollbar to move vertically if the whole canvas does not fit the window. If you use dumped labels in ordinary R plots, your plot must have the same dimensions as the `orditkplot` canvas to have identical location of the labels.

The function only displays one set of scores. However, you can use `ordipointlabel` to produce a result object that has different points and text types for several sets of scores and this can further edited fith `orditkplot`. For a good starting solution you need to scale the `ordipointlabel` result so that the points span over the whole horizontal axis.

The plot is a Tcl/Tk canvas, but the function tries to replicate standard graphical device of the platform, and it honours several graphical parameters (see `par`). Many of the graphical parameters can be given on the command line, and they will be passed to the function without influencing other graphical devices in R. At the moment, the following graphical parameters are honoured: `pch` `bg`, `cex`, `cex.axis`, `cex.lab`, `col` (for labels), `col.axis`, `col.lab`, `family` (for font faces), `fg`, `font`, `font.axis`, `font.lab`, `lheight`, `lwd` (for the box), `mar`, `mex`, `mgp`, `ps`, `tcl`. These can be set with `par`, and they also will influence other plots similarly.

The `tkcanvas` text cannot be rotated, and therefore vertical axis is not labelled, and `las` `par`ameter will not be honoured in the Tcl/Tk plot, but it will be honoured in the exported R plots and in `plot.orditkplot`.

Value

Function returns nothing useful directly, but you can save the edited graph to a file or dump the edited positions to an R session for further processing and plotting.

Note

You need **tcltk** package and R must have been configured with `capabilities` for `tcltk` when building the binary. Depending on your OS, you may need to start X11 and set the display before loading **tcltk** and starting the function (for instance, with `Sys.setenv("DISPLAY"=":0")`). See `tcltk-package`.

Author(s)

Jari Oksanen

See Also

Function `ordipointlabel` is an automatic procedure with similar goals of avoiding overplotting. See `ordiplot`, `plot.cca`, `ordirgl` and `orditorp` for alternative ordination plots, and `scores` for extracting ordination scores.

Examples

```
## The example needs user interaction and is not executed directly.
## It should work when pasted to the window.
## Not run:
data(varespec)
ord <- cca(varespec)
## Do something with the graph and end by clicking "Dismiss"
orditkplot(ord, mar = c(4,4,1,1)+.1, font=3)
## Use ordipointlabel to produce a plot that has both species and site
## scores in different colors and plotting symbols
pl <- ordipointlabel(ord)
orditkplot(pl)
## End(Not run)
```

`orditorp` *Add Text or Points to Ordination Plots*

Description

The function adds `text` or `points` to ordination plots. Text will be used if this can be done without overwriting other text labels, and points will be used otherwise. The function can help in reducing clutter in ordination graphics, but manual editing may still be necessary.

Usage

```
orditorp(x, display, labels, choices = c(1, 2), priority,
    cex = 0.7, pcex, col = par("col"), pcol,
    pch = par("pch"), air = 1, ...)
```

Arguments

`x`	A result object from ordination or an `ordiplot` result.
`display`	Items to be displayed in the plot. Only one alternative is allowed. Typically this is `"sites"` or `"species"`.
`labels`	Optional text used for labels. Row names will be used if this is missing.
`choices`	Axes shown.
`priority`	Text will be used for items with higher priority if labels overlap. This should be vector of the same length as the number of items plotted.
`cex, pcex`	Text and point sizes, see `plot.default`..
`col, pcol`	Text and point colours, see `plot.default`.
`pch`	Plotting character, see `points`.
`air`	Amount of empty space between text labels. Values <1 allow overlapping text.
`...`	Other arguments to `scores` (and its various methods), `text` and `points`.

Details

Function `orditorp` will add either text or points to an existing plot. The items with high `priority` will be added first and `text` will be used if this can be done without overwriting previous labels,and `points` will be used otherwise. If `priority` is missing, labels will be added from the outskirts to the centre. Function `orditorp` can be used with most ordination results, or plotting results from `ordiplot` or ordination plot functions (`plot.cca`, `plot.decorana`, `plot.metaMDS`).

Arguments can be passed to the relevant `scores` method for the ordination object (`x`) being drawn. See the relevant `scores` help page for arguments that can be used.

Value

The function returns invisibly a logical vector where `TRUE` means that item was labelled with text and `FALSE` means that it was marked with a point. The returned vector can be used as the `select` argument in ordination `text` and `points` functions.

Author(s)

Jari Oksanen

Examples

```
## A cluttered ordination plot :
data(BCI)
mod <- cca(BCI)
plot(mod, dis="sp", type="t")
```

```
# Now with orditorp and abbreviated species names
cnam <- make.cepnames(names(BCI))
plot(mod, dis="sp", type="n")
stems <- colSums(BCI)
orditorp(mod, "sp", label = cnam, priority=stems, pch="+", pcol="grey")
```

---

ordixyplot *Trellis (Lattice) Plots for Ordination*

Description

Functions `ordicloud`, `ordisplom` and `ordixyplot` provide an interface to plot ordination results using Trellis functions `cloud`, `splom` and `xyplot` in package **lattice**.

Usage

```
ordixyplot(x, data = NULL, formula, display = "sites", choices = 1:3,
    panel = "panel.ordi", aspect = "iso", envfit,
    type = c("p", "biplot"), ...)
ordisplom(x, data=NULL, formula = NULL,  display = "sites", choices = 1:3,
    panel = "panel.ordi", type = "p",  ...)
ordicloud(x, data = NULL, formula, display = "sites", choices = 1:3,
    panel = "panel.ordi3d", prepanel = "prepanel.ordi3d",  ...)
```

Arguments

`x`	An ordination result that `scores` knows: any ordination result in **vegan** and many others.
`data`	Optional data to amend ordination results. The ordination results are found from `x`, but you may give here data for other variables needed in plots. Typically these are environmental data.
`formula`	Formula to define the plots. A default formula will be used if this is omitted. The ordination axes must be called by the same names as in the ordination results (and these names vary among methods). In `ordisplom`, special character `.` refers to the ordination result.
`display`	The kind of scores: an argument passed to `scores`.
`choices`	The axes selected: an argument passed to `scores`.
`panel, prepanel`	The names of the panel and prepanel functions.
`aspect`	The aspect of the plot (passed to the **lattice** function).
`envfit`	Result of `envfit` function displayed in `ordixyplot`. Please note that this needs same `choices` as `ordixyplot`.
`type`	The type of plot. This knows the same alternatives as `panel.xyplot`. In addition `ordixyplot` has alternative `"biplot"` which displays fitted vectors and factor centroids of `envfit`, or in constrained ordination, the biplot arrows and factor centroids if `envfit` is not given.
`...`	Arguments passed to `scores` methods or **lattice** functions.

Details

The functions provide an interface to the corresponding **lattice** functions. All graphical parameters are passed to the **lattice** function so that these graphs are extremely configurable. See `Lattice` and `xyplot`, `splom` and `cloud` for details, usage and possibilities.

The argument `x` must always be an ordination result. The scores are extracted with **vegan** function `scores` so that these functions work with all **vegan** ordinations and many others.

The `formula` is used to define the models. All functions have simple default formulas which are used if `formula` is missing. If formula is omitted in `ordisplom` it produces a pairs plot of ordination axes and variables in `data`. If `formula` is given, ordination results must be referred to as `.` and other variables by their names. In other functions, the formula must use the names of ordination scores and names of `data`.

The ordination scores are found from `x`, and `data` is optional. The `data` should contain other variables than ordination scores to be used in plots. Typically, they are environmental variables (typically factors) to define panels or plot symbols.

The proper work is done by the panel function. The layout can be changed by defining own panel functions. See `panel.xyplot`, `panel.splom` and `panel.cloud` for details and survey of possibilities.

Ordination graphics should always be isometric: same scale should be used in all axes. This is controlled (and can be changed) with argument `aspect` in `ordixyplot`. In `ordicloud` the isometric scaling is defined in `panel` and `prepanel` functions. You must replace these functions if you want to have non-isometric scaling of graphs. You cannot select isometric scaling in `ordisplom`.

Value

The function return `Lattice` objects of class `"trellis"`.

Author(s)

Jari Oksanen

See Also

`Lattice`, `xyplot`, `splom`, `cloud`, `panel.splom`, `panel.cloud`

Examples

```
data(dune)
data(dune.env)
ord <- cca(dune)
## Pairs plots
ordisplom(ord)
ordisplom(ord, data=dune.env, choices=1:2)
ordisplom(ord, data=dune.env, form = ~ . | Management, groups=Manure)
## Scatter plot
ordixyplot(ord, data=dune.env, form = CA1 ~ CA2 | Management,
  groups=Manure)
## Choose a different scaling
```

```
ordixyplot(ord, scaling = 3)
## ... Slices of third axis
ordixyplot(ord, form = CA1 ~ CA2 | equal.count(CA3, 4), type = c("g","p"))
## Display environemntal variables
ordixyplot(ord, envfit = envfit(ord ~ Management + A1, dune.env, choices=1:3))
## 3D Scatter plots
ordicloud(ord, form = CA2 ~ CA3*CA1, groups = Manure, data = dune.env)
ordicloud(ord, form = CA2 ~ CA3*CA1 | Management, groups = Manure,
   data = dune.env, auto.key = TRUE, type = c("p","h"))
```

permCheck *Utility functions for permutation schemes*

Description

permCheck provides checking of permutation schemes for validity. numPerms calculates the maximum number of permutations possible under the current permutation scheme. allPerms enumerates all possible permutations for the given scheme. getNumObs is a utility function to return the number of observations for a range of R and ordination objects. permuplot produces a graphical representation of the selected permutation design.

Usage

```
permCheck(object, control = permControl(), make.all = TRUE)

## S3 method for class 'permCheck':
summary(object, ...)

numPerms(object, control = permControl())

allPerms(n, control = permControl(), max = 9999,
         observed = FALSE)

## S3 method for class 'allPerms':
summary(object, ...)

getNumObs(object, ...)

## Default S3 method:
getNumObs(object, ...)

## S3 method for class 'numeric':
getNumObs(object, ...)

## S3 method for class 'integer':
getNumObs(object, ...)

permuplot(n, control = permControl(), col = par("col"),
          hcol = "red", shade = "lightgrey", xlim = NULL, ylim = NULL,
          inset = 0.1, main = NULL, sub = NULL, ann = par("ann"),
          cex = par("cex"), ...)
```

Arguments

`object`	an R object. Specifically, for `getNumObs` any object handled by `scores`, data frames, matrices, and numeric and integer vectors. See Details for a complete description, especially for `numPerms`. For `summary.permCheck` an object of class `"permCheck"`. For `summary.allPerms` an object of class `"allPerms"`.
`control`	a list of control values describing properties of the permutation design, as returned by a call to `permControl`.
`make.all`	logical; should `permCheck` generate all possible permutations? Useful if want to check permutation design but not produce the matrix of all permutations.
`n`	the number of observations or an 'object' from which the number of observations can be determined via `getNumObs`.
`max`	the maximum number of permutations, below which complete enumeration will be attempted. See Details.
`observed`	logical, should the observed ordering of samples be returned as part of the complete enumeration? Default is `FALSE` to facilitate usage in higher level functions.
`col, xlim, ylim, main, sub, ann, cex`	Graphical parameters.
`hcol`	Colour to use for highlighting observations and the border colour of the polygons drawn when `type = "strata"`.
`shade`	The polygon shading colour (passed to argument `col` of function `polygon`) when `type = "strata"`.
`inset`	Proportion of range of x and y coordinates to add to the plot x and y limits. Used to create a bit of extra space around the margin of each plot.
`...`	arguments to other methods. For `permuplot` graphical parameters can be passed to plotting functions, though note that not all parameters will be accepted gracefully at the moment.

Details

`permCheck`, `allPerms`, `numPerms` and `permuplot` are utility functions for working with the new permutation schemes available in `permuted.index2`.

`permCheck` is used to check the current permutation schemes against the object to which it will be applied. It calculates the maximum number of possible permutations for the number of observations in `object` and the permutation scheme described by `control`. The returned object contains component `control`, an object of class `"permControl"` suitably modified if `permCheck` identifies a problem.

The main problem is requesting more permutations than possible with the number of observations and the permutation design. In such cases, `nperm` is reduced to equal the number of possible permutations, and complete enumeration of all permutations is turned on (`control$complete` is set to `TRUE`).

Alternatively, if the number of possible permutations is low, and less than `control$minperm`, it is better to enumerate all possible permutations, and as such complete enumeration of all permutations is turned on (`control$complete` is set to `TRUE`).

Function `numPerms` returns the number of permutations for the passed `object` and the selected permutation scheme. `object` can be one of a data frame, matrix, an object for which a scores method exists, or a numeric or integer vector. In the case of a numeric or integer vector, a vector of length 1 can be used and it will be expanded to a vector of length `object` (i.e., `1:object`) before computing the number of permutations. As such, `object` can be the number of observations not just the object containing the observations.

Function `allPerms` enumerates all possible permutations for the number of observations and the selected permutation scheme. It has `print` and `summary` methods. `allPerms` returns a matrix containing all possible permutations, possibly containing the observed ordering (if argument `observed` is `TRUE`). The rows of this matrix are the various permutations and the columns reflect the number of samples.

With free permutation designs, and restricted permutation schemes with large numbers of observations, there are a potentially huge number of possible permutations of the samples. It would be inefficient, not to mention incredibly time consuming, to enumerate them all. Storing all possible permutations would also become problematic in such cases. To control this and guard against trying to evaluate too large a number of permutations, if the number of possible permutations is larger than `max`, `allPerms` exits with an error.

Function `getNumObs` is a simple generic function to return the number of observations in a range of R objects. The default method will work for any object for which a `scores` method exists. This includes matrices and data frames, as well as specific methods for numeric or integer vectors.

`permuplot` is a graphical utility function, which produces a graphical representation of a permutation design. It takes the number of observations and an object returned by `permControl` as arguments and produces a plot on the currently active device. If strata are present in the design, the plotting region is split into sufficient plotting regions (one for each stratum), and the design in each stratum plotted.

Free permutation designs are represented by plotting the observation number at random x and y coordinates. Series designs (time series or line transects) are represented by plotting the observation numbers comprising the series in a circle and the start of the permuted series is highlighted using colour `hcol`. Grid designs are drawn on a regular grid and the top left observation in the original grid is highlighted using colour `hcol`. Note the ordering used is R's standard ordering for matrices - columns are filled first.

Value

For `permCheck` a list containing the maximum number of permutations possible and an object of class `"permControl"`.

For `allPerms`, and object of class `"allPerms"`, a matrix whose rows are the set of all possible permutations for the supplies number of observations and permutation scheme selected. The matrix has two additional attributes `control` and `observed`. Attribute `control` contains the argument `control` (possibly updated via `permCheck`). Attribute `observed` contains argument `observed`.

For `numPerms`, the (numeric) number of possible permutations.

For `getNumObs`, the (numeric) number of observations in `object`.

For `permuplot`, a plot on the currently active device.

Note

In general, mirroring `"series"` or `"grid"` designs doubles or quadruples, respectively,the number of permutations without mirroring (within levels of strata if present). This is **not** true in two special cases:

1. In `"grid"` designs where the number of columns is equal to 2, and
2. In `"series"` designs where the number of observations in a series is equal to 2.

For example, with 2 observations there are 2 permutations for `"series"` designs:

1. 1-2, and
2. 2-1.

If these two permutations were mirrored, we would have:

1. 2-1, and
2. 1-2.

It is immediately clear that this is the same set of permutations without mirroring (if one reorders the rows). A similar situation arises in `"grid"` designs where the number of **columns** per *grid* is equal to 2. Note that the number of rows per *grid* is not an issue here.

Author(s)

Gavin Simpson

See Also

`permuted.index2` and `permControl`.

Examples

```
## use example data from ?pyrifos
example(pyrifos)

## Demonstrate the maximum number of permutations for the pyrifos data
## under a series of permutation schemes

## no restrictions - lots of perms
(check1 <- permCheck(pyrifos, control = permControl(type = "free")))
summary(check1)

## no strata but data are series with no mirroring, so 132 permutations
permCheck(pyrifos, control = permControl(type = "series",
                   mirror = FALSE))
```

```
## no strata but data are series with mirroring, so 264 permutations
permCheck(pyrifos, control = permControl(type = "series",
                   mirror = TRUE))

## unrestricted within strata
permCheck(pyrifos, control = permControl(strata = ditch,
                   type = "free"))

## time series within strata, no mirroring
permCheck(pyrifos, control = permControl(strata = ditch,
                   type = "series", mirror = FALSE))

## time series within strata, with mirroring
permCheck(pyrifos, control = permControl(strata = ditch,
                   type = "series", mirror = TRUE))

## time series within strata, no mirroring, same permutation within strata
permCheck(pyrifos, control = permControl(strata = ditch,
                   type = "series", constant = TRUE))

## time series within strata, with mirroring, same permutation within strata
permCheck(pyrifos, control = permControl(strata = ditch,
                   type = "series", mirror = TRUE, constant = TRUE))

## permute strata
permCheck(pyrifos, permControl(strata = ditch, type = "free",
                               permute.strata = TRUE))

## this should also also for arbitrary vectors
vec1 <- permCheck(1:100)
vec2 <- permCheck(1:100, permControl())
all.equal(vec1, vec2)
vec3 <- permCheck(1:100, permControl(type = "series"))
all.equal(100, vec3$n)
vec4 <- permCheck(1:100, permControl(type = "series", mirror = TRUE))
all.equal(vec4$n, 200)

## enumerate all possible permutations
fac <- gl(2,6)
ctrl <- permControl(type = "grid", mirror = FALSE, strata = fac,
                    constant = TRUE, nrow = 3, ncol = 2)
numPerms(1:12, control = ctrl)
(tmp <- allPerms(12, control = ctrl, observed = TRUE))
(tmp2 <- allPerms(12, control = ctrl))
## turn on mirroring
ctrl$mirror <- TRUE
numPerms(1:12, control = ctrl)
(tmp3 <- allPerms(12, control = ctrl, observed = TRUE))
(tmp4 <- allPerms(12, control = ctrl))
## prints out details of the permutation scheme as
## well as the matrix of permutations
summary(tmp)
summary(tmp2)
```

```
## different numbers of observations per level of strata
fac <- factor(rep(1:3, times = c(3,2,2)))
## free permutations in levels of strata
numPerms(7, permControl(type = "free", strata = fac))
allPerms(7, permControl(type = "free", strata = fac))
## series permutations in levels of strata
numPerms(7, permControl(type = "series", strata = fac))
allPerms(7, permControl(type = "series", strata = fac))

## allPerms can work with a vector
vec <- c(3,4,5)
allPerms(vec)

## Tests for permuplot
n <- 25
## standard permutation designs
permuplot(n, permControl(type = "free"))
permuplot(n, permControl(type = "series"))
permuplot(n, permControl(type = "grid", nrow = 5, ncol = 5))

## restricted perms with mirroring
permuplot(n, permControl(type = "series", mirror = TRUE))
permuplot(n, permControl(type = "grid", nrow = 5, ncol = 5,
                             mirror = TRUE))

## perms within strata
fac <- gl(6, 20)
control <- permControl(type = "free", strata = fac)
permuplot(120, control = control, cex = 0.8)
control <- permControl(type = "series", strata = fac)
permuplot(120, control = control, cex = 0.8)
fac <- gl(6, 25)
control <- permControl(type = "grid", strata = fac,
                       nrow = 5, ncol = 5)
permuplot(150, control = control, cex = 0.8)

## perms within strata with mirroring
fac <- gl(6, 20)
control <- permControl(type = "series", strata = fac,
                       mirror = TRUE)
permuplot(120, control = control, cex = 0.8)
fac <- gl(6, 25)
control <- permControl(type = "grid", strata = fac,
                       nrow = 5, ncol = 5, mirror = TRUE)
permuplot(150, control = control, cex = 0.8)

## same perms within strata
fac <- gl(6, 20)
control <- permControl(type = "free", strata = fac,
                       constant = TRUE)
permuplot(120, control = control, cex = 0.8)
control <- permControl(type = "series", strata = fac,
                         constant = TRUE)
permuplot(120, control = control, cex = 0.8)
fac <- gl(6, 25)
control <- permControl(type = "grid", strata = fac,
                         nrow = 5, ncol = 5, constant = TRUE)
permuplot(150, control = control, cex = 0.8)

## same perms within strata with mirroring
fac <- gl(6, 20)
control <- permControl(type = "series", strata = fac,
                         mirror = TRUE, constant = TRUE)
permuplot(120, control = control, cex = 0.8)
fac <- gl(6, 25)
control <- permControl(type = "grid", strata = fac,
                         nrow = 5, ncol = 5, mirror = TRUE,
                         constant = TRUE)
permuplot(150, control = control, cex = 0.8)
```

permat — *Matrix Permutation Algorithms for Presence-Absence and Count Data*

Description

Individual (for count data) or incidence (for presence-absence data) based null models can be generated for community level simulations. Options for preserving characteristics of the original matrix (rows/columns sums, matrix fill) and restricted permutations (within strata based on spatial units, habitat classes or both) are discussed in the Details section. By using these functions, hypothesis testing can be separated from the null model generation, thus several tests might be applied on the same set of random matrices.

Usage

```
permatfull(m, fixedmar = "both", reg = NULL,
  hab = NULL, mtype = "count", times = 100)
permatswap(m, reg = NULL, hab = NULL, mtype = "count",
  method = "swap", times = 100, burnin = 10000, thin = 1000)
## S3 method for class 'permat':
plot(x, ...)
## S3 method for class 'permat':
summary(object, ...)
## S3 method for class 'summary.permat':
print(x, digits = 2, ...)
```

Arguments

`m`	a community data matrix with plots (samples) as rows and species (taxa) as columns.

`fixedmar`	character, stating which of the row/column sums should be preserved (`"none"`, `"rows"`, `"columns"`, `"both"`).
`reg`	numeric vector or factor with length same as `nrow(m)` for grouping rows within strata (regions) for restricted permutations. Unique values or levels are used.
`hab`	numeric vector or factor with length same as `nrow(m)` for grouping rows within strata (habitat classes) for restricted permutations. Unique values or levels are used.
`mtype`	matrix data type, either `"count"` for count data, or `"prab"` for presence-absence type incidence data.
`times`	number of permuted matrices.
`method`	character for method used for the swap algorithm (`"swap"`, `"tswap"`, `"backtrack"`) as described for function `commsimulator`. If `mtype="count"` only `"swap"` is available.
`burnin`	number of null communities discarded before proper analysis in sequential (`"swap"`, `"tswap"`) methods.
`thin`	number of discarded permuted matrices between two evaluations in sequential (`"swap"`, `"tswap"`) methods.
`x, object`	object of class `"permat"`
`digits`	number of digits used for rounding.
`...`	other arguments passed to methods.

Details

The function `permatfull` is useful when matrix fill is allowed to vary, and matrix type is `count`. The `fixedmar` argument is used to set constraints for permutation. If `none` of the margins are fixed, cells are randomised within the matrix. If `rows` or `columns` are fixed, cells within rows or columns are randomised, respectively. If `both` margins are fixed, the `r2dtable` function is used that is based on Patefield's (1981) algorithm. For presence absence data, matrix fill should be necessarily fixed, and `permatfull` is a wrapper for the function `commsimulator`. The `r00, r0, c0, quasiswap` algorithms of `commsimulator` are used for `"none"`, `"rows"`, `"columns"`, `"both"` values of the `fixedmar` argument, respectively

The function `permatswap` is useful when matrix fill (i.e. the proportion of empty cells) should be kept constant. `permatswap` uses different kinds of swap algorithms, and row and columns sums are fixed in all cases. For presence-absence data, the `swap` and `tswap` methods of `commsimulator` can be used. For count data, an experimental swap algorithm ('swapcount') is implemented that results in permuted matrices with fixed marginals and matrix fill at the same time. However, it seems that this model may not give true random matrices, and its use should be avoided in generating Null hypotheses. The code is provided only for methods comparisons, and may be removed from the future versions of **vegan**.

The 'swapcount' algorithm tries to find 2x2 submatrices (identified by 2 random row and 2 random column indices), that can be swapped in order to leave column and row totals and fill unchanged. First, the algorithm finds the largest value in the submatrix that can be swapped (d) and whether in diagonal or antidiagonal way. Submatrices that contain values larger than zero in either diagonal or antidiagonal position can be swapped. Swap means that the values in diagonal or antidiagonal positions are decreased by d, while remaining cells are increased by d. A swap is made only if fill doesn't change.

Constraints on row/colum sums, matrix fill, total sum and sums within strata can be checked by the `summary` method. `plot` method is for visually testing the randomness of the permuted matrices, especially for the swap algorithms. If there are any tendency in the graph, higher `burnin` and `thin` values can help.

Unrestricted and restricted permutations: if both `reg` and `hab` are `NULL`, functions perform unrestricted permutations. If either of the two is given, it is used as is for restricted permutations. If both are given, interaction is used for restricted permutations. Each strata should contain at least 2 rows in order to perform randomization (in case of low row numbers, swap algorithms can be rather slow).

Value

Functions `permatfull` and `permatswap` return an object of class `"permat"`.

`call`	the function call.
`orig`	the original data matrix used for permutations.
`perm`	a list of permuted matrices with length `times`.
`specs`	a list of other specifications (variable in length, depending on the function used): `reg`, `hab`, `burnin`, `thin`.

`summary.permat` returns a list containing mean Bray-Curtis dissimilarities calculated pairwise among original and permuted matrices, and check results of the constraints.

Author(s)

Péter Sólymos, ⟨solymos@ualberta.ca⟩; Jari Oksanen translated the original 'swapcount' algorithm for count data into C

References

Original references for presence-absence swap methods are given on help page of `commsimulator`.

Patefield, W. M. (1981) Algorithm AS159. An efficient method of generating r x c tables with given row and column totals. Applied Statistics 30, 91-97.

See Also

`commsimulator`, `r2dtable`, `sample`

Examples

```
## A simple artificial community data matrix.
m <- matrix(c(
   1,3,2,0,3,1,
   0,2,1,0,2,1,
   0,0,1,2,0,3,
   0,0,0,1,4,3
   ), 4, 6, byrow=TRUE)
```

```
## Using the swap algorithm to create a
## list of permuted matrices, where
## row/columns sums and matrix fill are preserved:
x1 <- permatswap(m, burnin = 1000, thin = 100)
summary(x1)
plot(x1)
## Unrestricted permutation retaining
## row/columns sums but not matrix fill:
x2 <- permatfull(m)
summary(x2)
plot(x2)
## Unrestricted permutation of presence-absence type
## not retaining row/columns sums:
x3 <- permatfull(m, "none", mtype="prab")
x3$orig  ## note: original matrix is binarized!
summary(x3)
## Restricted permutation,
## check sums within strata:
x4 <- permatfull(m, reg=c(1,1,2,2))
summary(x4)
```

`permuted.index2` *Unrestricted and restricted permutations*

Description

Unrestricted and restricted permutation designs for time series, line transects, spatial grids and blocking factors.

Usage

```
permuted.index2(n, control = permControl())

permControl(strata = NULL, nperm = 199, complete = FALSE,
            type = c("free", "series", "grid"),
            permute.strata = FALSE,
            maxperm = 9999, minperm = 99,
            mirror = FALSE, constant = FALSE,
            ncol = NULL, nrow = NULL,
            all.perms = NULL)

permute(i, n, control)
```

Arguments

`n`	numeric; the length of the returned vector of permuted values. Usually the number of observations under consideration.
`control`	a list of control values describing properties of the permutation design, as returned by a call to `permControl`.

`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
`nperm`	the number of permutations.
`complete`	logical; should complete enumeration of all permutations be performed?
`type`	the type of permutations required. One of `"free"`, `"series"`, or `"grid"`. See Details.
`permute.strata`	logical; should strata be permuted? See Details.
`maxperm`	the maximum number of permutations to perform. Currently unused.
`minperm`	the lower limit to the number of possible permutations at which complete enumeration is performed. See argument `complete` and Details, below.
`mirror`	logical; should mirroring of sequences be allowed?
`constant`	logical; should the same permutation be used within each level of strata? If `FALSE` a separate, possibly restricted, permutation is produced for each level of `strata`.
`ncol, nrow`	numeric; the number of columns and rows of samples in the spatial grid respectiavly.
`all.perms`	an object of class `allPerms`, the result of a call to `allPerms`.
`i`	integer; row of `control$all.perms` to return.

Details

`permuted.index2` can generate permutations for a wide range of restricted permutation schemes. A small selection of the available combinations of options is provided in the Examples section below.

Argument `mirror` determines whether grid or series permutations can be mirrored. Consider the sequence 1,2,3,4. The relationship between consecutive observations is preserved if we reverse the sequence to 4,3,2,1. If there is no inherent direction in your experimental design, mirrored permutations can be considered part of the Null model, and as such increase the number of possible permutations. The default is to not use mirroring so you must explicitly turn this on using `mirror = TRUE` in `permControl`.

To permute `strata` rather than the observations within the levels of `strata`, use `permute.strata = TRUE`. However, note that the number of observations within each level of strata **must** be equal!

For some experiments, such as BACI designs, one might wish to use the same permutation within each level of strata. This is controlled by argument `constant`. If `constant = TRUE` then the same permutation will be generated for each level of `strata`. The default is `constant = FALSE`.

`permute` is a higher level utility function for use in a loop within a function implementing a permutation test. The main purpose of `permute` is to return the correct permutation in each iteration of the loop, either a random permutation from the current design or the next permutation from `control$all.perms` if it is not `NULL` and `control$complete` is `TRUE`.

Value

For `permuted.index2` a vector of length `n` containing a permutation of the observations 1, . . . , n using the permutation scheme described by argument `control`.

For `permControl` a list with components for each of the possible arguments.

Note

`permuted.index2` is currently used in one Vegan function; `permutest.betadisper`. Over time, the other functions that currently use the older `permuted.index` will be updated to use `permuted.index2`.

Author(s)

Gavin Simpson

See Also

`permCheck`, a utility function for checking permutation scheme described by `permControl`.

Examples

```
set.seed(1234)

## unrestricted permutations
permuted.index2(20)

## observations represent a time series of line transect
permuted.index2(20, control = permControl(type = "series"))

## observations represent a time series of line transect
## but with mirroring allowed
permuted.index2(20, control = permControl(type = "series", mirror = TRUE))

## observations represent a spatial grid
perms <- permuted.index2(20, permControl(type = "grid",
                                         ncol = 4, nrow = 5))
## view the permutation as a grid
matrix(matrix(1:20, nrow = 5, ncol = 4)[perms], ncol = 4, nrow = 5)

## random permutations in presence of strata
block <- gl(4, 5)
permuted.index2(20, permControl(strata = block, type = "free"))
## as above but same random permutation within strata
permuted.index2(20, permControl(strata = block, type = "free",
                                constant = TRUE))

## time series within each level of block
permuted.index2(20, permControl(strata = block, type = "series"))
## as above, but  with same permutation for each level
permuted.index2(20, permControl(strata = block, type = "series",
                                constant = TRUE))
```

```
## spatial grids within each level of block
permuted.index2(100, permControl(strata = block, type = "grid",
                                 ncol = 5, nrow = 5))
## as above, but with same permutation for each level
permuted.index2(100, permControl(strata = block, type = "grid",
                                 ncol = 5, nrow = 5, constant = TRUE))

## permuting levels of block instead of observations
permuted.index2(20, permControl(strata = block, type = "free",
                                permute.strata = TRUE))

## Simple function using permute() to assess significance
## of a t.test
pt.test <- function(x, group, control) {
    ## function to calculate t
    t.statistic <- function(x, y) {
        m <- length(x)
        n <- length(y)
        ## means and variances, but for speed
        xbar <- .Internal(mean(x))
        ybar <- .Internal(mean(y))
        xvar <- .Internal(cov(x, NULL, 1, FALSE))
        yvar <- .Internal(cov(y, NULL, 1, FALSE))
        pooled <- sqrt(((m-1)*xvar + (n-1)*yvar) / (m+n-2))
        (xbar - ybar) / (pooled * sqrt(1/m + 1/n))
    }
    ## check the control object
    control <- permCheck(x, control)$control
    ## number of observations
    nobs <- getNumObs(x)
    ## group names
    lev <- names(table(group))
    ## vector to hold results, +1 because of observed t
    t.permu <- numeric(length = control$nperm) + 1
    ## calculate observed t
    t.permu[1] <- t.statistic(x[group == lev[1]], x[group == lev[2]])
    ## generate randomisation distribution of t
    for(i in seq_along(t.permu)) {
        ## return a permutation
        want <- permute(i, nobs, control)
        ## calculate permuted t
        t.permu[i+1] <- t.statistic(x[want][group == lev[1]],
                                    x[want][group == lev[2]])
    }
    ## pval from permutation test
    pval <- sum(abs(t.permu) >= abs(t.permu[1])) / (control$nperm + 1)
    ## return value
    return(list(t.stat = t.permu[1], pval = pval))
}

## generate some data with slightly different means
set.seed(1234)
```

```
gr1 <- rnorm(20, mean = 9)
gr2 <- rnorm(20, mean = 10)
dat <- c(gr1, gr2)
## grouping variable
grp <- gl(2, 20, labels = paste("Group", 1:2))
## create the permutation design
control <- permControl(type = "free", nperm = 999)
## perform permutation t test
perm.val <- pt.test(dat, grp, control)
perm.val

## compare perm.val with the p-value from t.test()
t.test(dat ~ grp, var.equal = TRUE)
```

`permutest.betadisper` *Permutation test of ultivariate homogeneity of groups dispersions (variances)*

Description

Implements a permutation-based test of multivariate homogeneity of group dispersions (variances) for the results of a call to `betadisper`.

Usage

```
## S3 method for class 'betadisper':
permutest(x, pairwise = FALSE,
          control = permControl(nperm = 999), ...)
```

Arguments

`x`	an object of class `"betadisper"`, the result of a call to `betadisper`.
`pairwise`	logical; perform pairwise comparisons of group means?
`control`	a list of control values for the permutations to replace the default values returned by the function `permControl`
`...`	Arguments passed to other methods.

Details

To test if one or more groups is more variable than the others, ANOVA of the distances to group centroids can be performed and parametric theory used to interpret the significance of F. An alternative is to use a permutation test. `permutest.betadisper` permutes model residuals to generate a permutation distribution of F under the Null hypothesis of no difference in dispersion between groups.

Pairwise comprisons of group mean dispersions can be performed by setting argument `pairwise` to `TRUE`. A classicial t test is performed on the pairwise group dispersions. This is combined with a permutation test based on the t statistic calculated on pariwise group dispersions. An alternative to the classical comparison of group dispersions, is to calculate Tukey's Honest Significant Differences between groups, via `TukeyHSD.betadisper`.

Value

`permutest.betadisper` returns a list of class `"permutest.betadisper"` with the following components:

`tab`	the ANOVA table which is an object inheriting from class `"data.frame"`.
`pairwise`	a list with components `observed` and `permuted` containing the observed and permuted p-values for pairwise comparisons of group mean distances (dispersions or variances).
`groups`	character; the levels of the grouping factor.
`control`	a list, the result of a call to `permControl`.

Author(s)

Gavin L. Simpson

References

Anderson, M.J. (2006) Distance-based tests for homogeneity of multivariate dispersions. *Biometrics* **62(1)**, 245–253.

Anderson, M.J., Ellingsen, K.E. & McArdle, B.H. (2006) Multivariate dispersion as a measure of beta diversity. *Ecology Letters* **9(6)**, 683–693.

See Also

For the main fitting function see `betadisper`. For an alternative approach to determining which groups are more variable, see `TukeyHSD.betadisper`.

Examples

```
data(varespec)

## Bray-Curtis distances between samples
dis <- vegdist(varespec)

## First 16 sites grazed, remaining 8 sites ungrazed
groups <- factor(c(rep(1,16), rep(2,8)), labels = c("grazed","ungrazed"))

## Calculate multivariate dispersions
mod <- betadisper(dis, groups)
mod

## Perform test
anova(mod)

## Permutation test for F
```

```
permutest(mod, pairwise = TRUE)

## Tukey's Honest Significant Differences
(mod.HSD <- TukeyHSD(mod))
plot(mod.HSD)
```

`plot.cca` *Plot or Extract Results of Constrained Correspondence Analysis or Redundancy Analysis*

Description

Functions to plot or extract results of constrained correspondence analysis (`cca`), redundancy analysis (`rda`) or constrained analysis of principal coordinates (`capscale`).

Usage

```
## S3 method for class 'cca':
plot(x, choices = c(1, 2), display = c("sp", "wa", "cn"),
         scaling = 2, type, xlim, ylim, const, ...)
## S3 method for class 'cca':
text(x, display = "sites", labels, choices = c(1, 2), scaling = 2,
    arrow.mul, head.arrow = 0.05, select, ...)
## S3 method for class 'cca':
points(x, display = "sites", choices = c(1, 2), scaling = 2,
    arrow.mul, head.arrow = 0.05, select, ...)
## S3 method for class 'cca':
scores(x, choices=c(1,2), display=c("sp","wa","cn"), scaling=2, ...)
## S3 method for class 'rda':
scores(x, choices=c(1,2), display=c("sp","wa","cn"), scaling=2,
    const, ...)
## S3 method for class 'cca':
summary(object, scaling = 2, axes = 6, display = c("sp", "wa",
    "lc", "bp", "cn"), digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'summary.cca':
print(x, digits = x$digits, head = NA, tail = head, ...)
## S3 method for class 'summary.cca':
head(x, n = 6, tail = 0, ...)
## S3 method for class 'summary.cca':
tail(x, n = 6, head = 0, ...)
```

Arguments

`x, object`	A `cca` result object.
`choices`	Axes shown.

`display`	Scores shown. These must some of the alternatives `species` or `sp` for species scores, `sites` or `wa` for site scores, `lc` for linear constraints or "LC scores", or `bp` for biplot arrows or `cn` for centroids of factor constraints instead of an arrow.
`scaling`	Scaling for species and site scores. Either species (`2`) or site (`1`) scores are scaled by eigenvalues, and the other set of scores is left unscaled, or with `3` both are scaled symmetrically by square root of eigenvalues. Corresponding negative values can be used in `cca` to additionally multiply results with $\sqrt{(}1/(1-\lambda))$. This scaling is know as Hill scaling (although it has nothing to do with Hill's rescaling of `decorana`). With corresponding negative values in`rda`, species scores are divided by standard deviation of each species and multiplied with an equalizing constant. Unscaled raw scores stored in the result can be accessed with `scaling = 0`.
`type`	Type of plot: partial match to `text` for text labels, `points` for points, and `none` for setting frames only. If omitted, `text` is selected for smaller data sets, and `points` for larger.
`xlim, ylim`	the x and y limits (min,max) of the plot.
`labels`	Optional text to be used instead of row names.
`arrow.mul`	Factor to expand arrows in the graph. Arrows will be scaled automatically to fit the graph if this is missing.
`head.arrow`	Default length of arrow heads.
`select`	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices of displayed items.
`const`	General scaling constant to `rda` scores. The default is to use constant to give biplot scores, or scores that approximate original data.
`axes`	Number of axes in summaries.
`digits`	Number of digits in output.
`n, head, tail`	Number of rows printed from the head and tail of species and site scores. Default `NA` prints all.
`...`	Parameters passed to other functions.

Details

Same `plot` function will be used for `cca` and `rda`. This produces a quick, standard plot with current `scaling`.

The `plot` function sets colours (`col`), plotting characters (`pch`) and character sizes (`cex`) to certain standard values. For a fuller control of produced plot, it is best to call `plot` with `type="none"` first, and then add each plotting item separately using `text.cca` or `points.cca` functions. These use the default settings of standard `text` and `points` functions and accept all their parameters, allowing a full user control of produced plots.

Environmental variables receive a special treatment. With `display="bp"`, arrows will be drawn. These are labelled with `text` and unlabelled with `points`. The basic `plot` function uses a simple (but not very clever) heuristics for adjusting arrow lengths to plots, but the user can give the expansion factor in `mul.arrow`. With `display="cn"` the centroids of levels of `factor` variables are displayed (these are available only if there were factors and a formula interface was used in `cca` or `rda`). With this option continuous variables still are presented as arrows and ordered factors as arrows and centroids.

If you want to have still a better control of plots, it is better to produce them using primitive `plot` commands. Function `scores` helps in extracting the needed components with the selected `scaling`.

Function `summary` lists all scores and the output can be very long. You can suppress scores by setting `axes = 0` or `display = NA` or `display = NULL`. You can display some first or last (or both) rows of scores by using `head` or `tail` or explicit `print` command for the `summary`.

Palmer (1993) suggested using linear constraints ("LC scores") in ordination diagrams, because these gave better results in simulations and site scores ("WA scores") are a step from constrained to unconstrained analysis. However, McCune (1997) showed that noisy environmental variables (and all environmental measurements are noisy) destroy "LC scores" whereas "WA scores" were little affected. Therefore the `plot` function uses site scores ("WA scores") as the default. This is consistent with the usage in statistics and other functions in R (`lda`, `cancor`).

Value

The `plot` function returns invisibly a plotting structure which can be used by function `identify.ordiplot` to identify the points or other functions in the `ordiplot` family.

Note

Package **ade4** has function `cca` which returns constrained correspondence analysis of the same class as the **vegan** function. If you have results of **ade4** in your working environment, **vegan** functions may try to handle them and fail with cryptic error messages. However, there is a simple utility function `ade2vegancca` which tries to translate **ade4** `cca` results to **vegan** `cca` results so that some **vegan** functions may work partially with **ade4** objects (with a warning).

Author(s)

Jari Oksanen

See Also

`cca`, `rda` and `capscale` for getting something to plot, `ordiplot` for an alternative plotting routine and more support functions, and `text`, `points` and `arrows` for the basic routines.

Examples

```
data(dune)
data(dune.env)
mod <- cca(dune ~ A1 + Moisture + Management, dune.env)
plot(mod, type="n")
text(mod, dis="cn")
points(mod, pch=21, col="red", bg="yellow", cex=1.2)
text(mod, "species", col="blue", cex=0.8)
## Limited output of 'summary'
head(summary(mod), tail=2)
```

prc — *Principal Response Curves for Treatments with Repeated Observations*

Description

Principal Response Curves (PRC) are a special case of Redundancy Analysis (`rda`) for multivariate responses in repeated observation design. They were originally suggested for ecological communities. They should be easier to interpret than traditional constrained ordination.

Usage

```
prc(response, treatment, time, ...)
## S3 method for class 'prc':
summary(object, axis = 1, scaling = 2, digits = 4, ...)
## S3 method for class 'prc':
plot(x, species = TRUE, select, scaling = 2, axis = 1, type = "l",
    xlab, ylab, ylim, lty = 1:5, col = 1:6, pch, legpos, cex = 0.8,
    ...)
```

Arguments

`response`	Multivariate response data. Typically these are community (species) data. If the data are counts, they probably should be log transformed prior to the analysis.
`treatment`	A factor for treatments.
`time`	An unordered factor defining the observations times in the repeated design.
`object, x`	An `prc` result object.
`axis`	Axis shown (only one axis can be selected).
`scaling`	Scaling of species scores, identical to the `scaling` in `scores.rda`.
`digits`	Number of significant digits displayed.
`species`	Display species scores.
`select`	Vector to select displayed species. This can be a vector of indices or a logical vector which is `TRUE` for the selected species
`type`	Type of plot: `"l"` for lines, `"p"` for points or `"b"` for both.
`xlab, ylab`	Text to replace default axis labels.
`ylim`	Limits for the vertical axis.
`lty, col, pch`	Line type, colour and plotting characters (defaults supplied).
`legpos`	The position of the `legend`. A guess is made if this is not supplied, and `NA` will suppress legend.
`cex`	Character expansion for symbols and species labels.
`...`	Other parameters passed to functions.

Details

PRC is a special case of `rda` with a single factor for `treatment` and a single factor for `time` points in repeated observations. In **vegan**, the corresponding `rda` model is defined as `rda(response ~ treatment * time + Condition(time))`. Since the `time` appears twice in the model formula, its main effects will be aliased, and only interaction terms are available, and will be used in PRC. Instead of usual multivariate ordination diagrams, PRC uses canonical (regression) coefficients and species scores for a single axis. All that the current functions do is to provide a special `summary` and `plot` methods that display the `rda` results in the PRC fashion. With default contrasts (`contr.treatment`) the coefficients are contrasts against the first level, and the levels must be arranged so that the first level is the control.

Function `summary` prints the species scores and the coefficients. Function `plot` plots coefficients against `time` using `matplot`, and has similar defaults. The graph (and PRC) is meaningful only if the first `treatment` level is the control, as the results are contrasts to the first level when unordered factors are used. The plot also displays species scores on the right vertical axis using function `linestack`. Typically the number of species is so high that not all can be displayed with the default settings, but users can reduce character size or padding (`air`) in `linestack`, or `select` only a subset of the species. A legend will be displayed unless suppressed with `legpos = NA`, and the functions tries to guess where to put the legend if `legpos` is not supplied.

Value

The function is a special case of `rda` and returns its result object (see `cca.object`). However, a special `summary` and `plot` methods display returns differently than in `rda`.

Warning

The first level of `treatment` must be the control, and the treatment contrasts must be used (see `contr.treatment`), so that results are expressed as contrasts to the control. The function works with other contrast types also, but then the user must take care that the resulting coefficients are meaningful. The `time` must be an unordered factor.

Author(s)

Jari Oksanen

References

van den Brink, P.J. & ter Braak, C.J.F. (1999). Principal response curves: Analysis of time-dependent multivariate responses of biological community to stress. Environmental Toxicology and Chemistry, 18, 138–148.

See Also

`rda`, `anova.cca`.

Examples

```
# Chlorpyrifos experiment and experimental design
data(pyrifos)
week <- gl(11, 12, labels=c(-4, -1, 0.1, 1, 2, 4, 8, 12, 15, 19, 24))
dose <- factor(rep(c(0.1, 0, 0, 0.9, 0, 44, 6, 0.1, 44, 0.9, 0, 6), 11))
# PRC
mod <- prc(pyrifos, dose, week)
mod            # RDA
summary(mod)   # PRC
logabu <- colSums(pyrifos)
plot(mod, select = logabu > 100)
# Permutations should be done only within one week, and we only
# are interested on the first axis
anova(mod, strata = week, first=TRUE)
```

`predict.cca` *Prediction Tools for [Constrained] Ordination (CCA, RDA, DCA, CA, PCA)*

Description

Function `predict` can be used to find site and species scores with new data sets.

Usage

```
## S3 method for class 'cca':
fitted(object, model = c("CCA", "CA"),
    type =  c("response", "working"), ...)
## S3 method for class 'cca':
predict(object, newdata, type = c("response", "wa", "sp", "lc"),
    rank = "full", model = c("CCA", "CA"), scaling = FALSE, ...)
calibrate.cca(object, newdata, rank = "full", ...)
## S3 method for class 'cca':
coef(object, ...)
## S3 method for class 'decorana':
predict(object, newdata, type = c("response", "sites", "species"),
    rank = 4, ...)
```

Arguments

`object`	A result object from `cca`, `rda`, `capscale` or `decorana`.
`model`	Show constrained (`"CCA"`) or unconstrained (`"CA"`) results.
`newdata`	New data frame to be used in prediction of species and site scores or for calibration. Usually this a new community data frame, but for `predict.cca type = "lc"` it must be an environment data frame, and for `type = "response"` this is ignored.

`type`	The type of prediction, fitted values or residuals: In `fitted` and `residuals`, `"response"` scales results so that the same ordination gives the same results, and `"working"` gives the values used internally, that is after Chi-square standardization in `cca` and scaling and centring in `rda`. In `predict` `"response"` gives an approximation of the original data matrix, `"wa"` the site scores as weighted averages of the community data, `"lc"` the site scores as linear combinations of environmental data, and `"sp"` the species scores. In `predict.decorana` the alternatives are scores for `"sites"` or `"species"`.
`rank`	The rank or the number of axes used in the approximation. The default is to use all axes (full rank) of the `"model"` or all available four axes in `predict.decorana`.
`scaling`	Scaling or predicted scores with the same meaning as in `cca`, `rda` and `capscale`.
`...`	Other parameters to the functions.

Details

Function `fitted` gives the approximation of the original data matrix from the ordination result either in the scale of the response or as scaled internally by the function. Function `residuals` gives the approximation of the original data from the unconstrained ordination. With argument `type = "response"` the `fitted.cca` and `residuals.cca` function both give the same marginal totals as the original data matrix, and their entries do not add up to the original data. They are defined so that for model `mod <- cca(y ~ x)`, `cca(fitted(mod))` is equal to constrained ordination, and `cca(residuals(mod))` is equal to unconstrained part of the ordination.

Function `predict` can find the estimate of the original data matrix (`type = "response"`) with any rank. With `rank = "full"` it is identical to `fitted`. In addition, the function can find the species scores or site scores from the community data matrix. The function can be used with new data, and it can be used to add new species or site scores to existing ordinations. The function returns (weighted) orthonormal scores by default, and you must specify explicit `scaling` to add those scores to ordination diagrams. With `type = "wa"` the function finds the site scores from species scores. In that case, the new data can contain new sites, but species must match in the original and new data. With `type = "sp"` the function finds species scores from site constraints (linear combination scores). In that case the new data can contain new species, but sites must match in the original and new data. With `type = "lc"` the function finds the linear combination scores for sites from environmental data. In that case the new data frame must contain all constraining and conditioning environmental variables of the model formula. If a completely new data frame is created, extreme care is needed defining variables similarly as in the original model, in particular with (ordered) factors. If ordination was performed with the formula interface, the `newdata` also can be a data frame or matrix, but extreme care is needed that the columns match in the original and `newdata`.

Function `calibrate.cca` finds estimates of constraints from community ordination or `"wa"` scores from `cca`, `rda` and `capscale`. This is often known as calibration, bioindication or environmental reconstruction. Basically, the method is similar to projecting site scores onto biplot arrows, but it uses regression coefficients. The function can be called with `newdata` so that cross-validation is possible. The `newdata` may contain new sites, but species must match in the original and new data The function does not work with ‘partial’ models with `Condition` term, and it cannot be used with `newdata` for `capscale` results. The results may only be interpretable for continuous variables.

Function `coef` will give the regression coefficients from centred environmental variables (constraints and conditions) to linear combination scores. The coefficients are for unstandardized environmental variables. The coefficients will be `NA` for aliased effects.

Function `predict.decorana` is similar to `predict.cca`. However, `type = "species"` is not available in detrended correspondence analysis (DCA), because detrending destroys the mutual reciprocal averaging (except for the first axis when rescaling is not used). Detrended CA does not attempt to approximate the original data matrix, so `type = "response"` has no meaning in detrended analysis (except with `rank = 1`).

Value

The functions return matrices or vectors as is appropriate.

Author(s)

Jari Oksanen.

References

Greenacre, M. J. (1984). Theory and applications of correspondence analysis. Academic Press, London.

See Also

`cca`, `rda`, `capscale`, `decorana`, `vif`, `goodness.cca`.

Examples

```
data(dune)
data(dune.env)
mod <- cca(dune ~ A1 + Management + Condition(Moisture), data=dune.env)
# Definition of the concepts 'fitted' and 'residuals'
mod
cca(fitted(mod))
cca(residuals(mod))
# Remove rare species (freq==1) from 'cca' and find their scores
# 'passively'.
freq <- specnumber(dune, MARGIN=2)
freq
mod <- cca(dune[, freq>1] ~ A1 + Management + Condition(Moisture), dune.env)
predict(mod, type="sp", newdata=dune[, freq==1], scaling=2)
# New sites
predict(mod, type="lc", new=data.frame(A1 = 3, Management="NM", Moisture="2"), scal=2)
# Calibration and residual plot
mod <- cca(dune ~ A1 + Moisture, dune.env)
pred <- calibrate.cca(mod)
pred
with(dune.env, plot(A1, pred[,"A1"] - A1, ylab="Prediction Error"))
abline(h=0)
```

procrustes *Procrustes Rotation of Two Configurations and PROTEST*

Description

Function `procrustes` rotates a configuration to maximum similarity with another configuration. Function `protest` tests the non-randomness ('significance') between two configurations.

Usage

```
procrustes(X, Y, scale = TRUE, symmetric = FALSE, scores = "sites", ...)
## S3 method for class 'procrustes':
summary(object, digits = getOption("digits"), ...)
## S3 method for class 'procrustes':
plot(x, kind=1, choices=c(1,2), xlab, ylab, main,
     ar.col = "blue", len=0.05, ...)
## S3 method for class 'procrustes':
points(x, display = c("target", "rotated"), ...)
## S3 method for class 'procrustes':
lines(x, type = c("segments", "arrows"), choices = c(1, 2), ...)
## S3 method for class 'procrustes':
residuals(object, ...)
## S3 method for class 'procrustes':
fitted(object, truemean = TRUE, ...)
protest(X, Y, scores = "sites", permutations = 1000, strata, ...)
```

Arguments

`X`	Target matrix
`Y`	Matrix to be rotated.
`scale`	Allow scaling of axes of `Y`.
`symmetric`	Use symmetric Procrustes statistic (the rotation will still be non-symmetric).
`scores`	Kind of scores used. This is the `display` argument used with the corresponding `scores` function: see `scores`, `scores.cca` and `scores.cca` for alternatives.
`x, object`	An object of class `procrustes`.
`digits`	Number of digits in the output.
`kind`	For `plot` function, the kind of plot produced: `kind = 1` plots shifts in two configurations, `kind = 0` draws a corresponding empty plot, and `kind = 2` plots an impulse diagram of residuals.
`choices`	Axes (dimensions) plotted.
`xlab, ylab`	Axis labels, if defaults unacceptable.
`main`	Plot title, if default unacceptable.
`display`	Show only the `"target"` or `"rotated"` matrix as points.

`type`	Combine `target` and `rotated` points with line segments or arrows.
`truemean`	Use the original range of target matrix instead of centring the fitted values.
`permutations`	Number of permutation to assess the significance of the symmetric Procrustes statistic.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
`ar.col`	Arrow colour.
`len`	Width of the arrow head.
`...`	Other parameters passed to functions. In `procrustes` and `protest` parameters are passed to `scores`, in graphical functions to underlying graphical functions.

Details

Procrustes rotation rotates a matrix to maximum similarity with a target matrix minimizing sum of squared differences. Procrustes rotation is typically used in comparison of ordination results. It is particularly useful in comparing alternative solutions in multidimensional scaling. If `scale=FALSE`, the function only rotates matrix `Y`. If `scale=TRUE`, it scales linearly configuration `Y` for maximum similarity. Since `Y` is scaled to fit `X`, the scaling is non-symmetric. However, with `symmetric=TRUE`, the configurations are scaled to equal dispersions and a symmetric version of the Procrustes statistic is computed.

Instead of matrix, `X` and `Y` can be results from an ordination from which `scores` can extract results. Function `procrustes` passes extra arguments to `scores`, `scores.cca` etc. so that you can specify arguments such as `scaling`.

Function `plot` plots a `procrustes` object and returns invisibly an `ordiplot` object so that function `identify.ordiplot` can be used for identifying points. The items in the `ordiplot` object are called `heads` and `points` with `kind=1` (ordination diagram) and `sites` with `kind=2` (residuals). In ordination diagrams, the arrow heads point to the target configuration, which may be either logical or illogical. Target and original rotated axes are shown as cross hairs in two-dimensional Procrustes analysis, and with a higher number of dimensions, the rotated axes are projected onto plot with their scaled and centred range. Function `plot` passes parameters to underlying plotting functions. For full control of plots, you can draw the axes using `plot` with `kind = 0`, and then add items with `points` or `lines`. These functions pass all parameters to the underlying functions so that you can select the plotting characters, their size, colours etc., or you can select the width, colour and type of line `segments` or arrows, or you can select the orientation and head width of `arrows`.

Function `residuals` returns the pointwise residuals, and `fitted` the fitted values, either centred to zero mean (if `truemean=FALSE`) or with the original scale (these hardly make sense if `symmetric = TRUE`). In addition, there are `summary` and `print` methods.

If matrix `X` has a lower number of columns than matrix `Y`, then matrix `X` will be filled with zero columns to match dimensions. This means that the function can be used to rotate an ordination configuration to an environmental variable (most practically extracting the result with the `fitted` function).

Function `protest` calls `procrustes(..., symmetric = TRUE)` repeatedly to estimate the 'significance' of the Procrustes statistic. Function `protest` uses a correlation-like statistic derived from the symmetric Procrustes sum of squares ss as $r = \sqrt{(1 - ss)}$, and sometimes called m_{12}. Function `protest` has own `print` method, but otherwise uses `procrustes` methods. Thus `plot` with a `protest` object yields a "Procrustean superimposition plot."

Value

Function `procrustes` returns an object of class `procrustes` with items. Function `protest` inherits from `procrustes`, but amends that with some new items:

`Yrot`	Rotated matrix `Y`.
`X`	Target matrix.
`ss`	Sum of squared differences between `X` and `Yrot`.
`rotation`	Orthogonal rotation matrix.
`translation`	Translation of the origin.
`scale`	Scaling factor.
`symmetric`	Type of `ss` statistic.
`call`	Function call.
`t0`	This and the following items are only in class `protest`: Procrustes correlation from non-permuted solution.
`t`	Procrustes correlations from permutations.
`signif`	'Significance' of `t`
`permutations`	Number of permutations.
`strata`	The name of the stratifying variable.
`stratum.values`	Values of the stratifying variable.

Note

The function `protest` follows Peres-Neto & Jackson (2001), but the implementation is still after Mardia *et al.* (1979).

Author(s)

Jari Oksanen

References

Mardia, K.V., Kent, J.T. and Bibby, J.M. (1979). *Multivariate Analysis*. Academic Press.

Peres-Neto, P.R. and Jackson, D.A. (2001). How well do multivariate data sets match? The advantages of a Procrustean superimposition approach over the Mantel test. *Oecologia* 129: 169-178.

See Also

`isoMDS`, `initMDS` for obtaining objects for `procrustes`, and `mantel` for an alternative to `protest` without need of dimension reduction.

Examples

```
data(varespec)
vare.dist <- vegdist(wisconsin(varespec))
library(MASS)  ## isoMDS
mds.null <- isoMDS(vare.dist, tol=1e-7)
mds.alt <- isoMDS(vare.dist, initMDS(vare.dist), maxit=200, tol=1e-7)
vare.proc <- procrustes(mds.alt, mds.null)
vare.proc
summary(vare.proc)
plot(vare.proc)
plot(vare.proc, kind=2)
residuals(vare.proc)
```

`pyrifos` *Response of Aquatic Invertebrates to Insecticide Treatment*

Description

The data are log transformed abundances of aquatic invertebrate in twelve ditches studied in eleven times before and after an insecticide treatment.

Usage

```
data(pyrifos)
```

Format

A data frame with 132 observations on the log-transformed abundances of 178 species. There are only twelve sites (ditches, mesocosms), but these were studied repeatedly in eleven occasions. The treatment levels, treatment times, or ditch ID's are not in the data frame, but the data are very regular, and the example below shows how to obtain these external variables.

Details

This data set was obtained from an experiment in outdoor experimental ditches. Twelve mesocosms were allocated at random to treatments; four served as controls, and the remaining eight were treated once with the insecticide chlorpyrifos, with nominal dose levels of 0.1, 0.9, 6, and 44 μg/ L in two mesocosms each. The example data set invertebrates. Sampling was done 11 times, from week -4 pre-treatment through week 24 post-treatment, giving a total of 132 samples (12 mesocosms times 11 sampling dates), see van den Brink & ter Braak (1999) for details. The data set contains only the species data, but the example below shows how to obtain the treatment, time and ditch ID variables.

Source

CANOCO 4 example data, with the permission of Cajo J. F. ter Braak.

References

van den Brink, P.J. & ter Braak, C.J.F. (1999). Principal response curves: Analysis of time-dependent multivariate responses of biological community to stress. Environmental Toxicology and Chemistry, 18, 138–148.

Examples

```
data(pyrifos)
ditch <- gl(12, 1, length=132)
week <- gl(11, 12, labels=c(-4, -1, 0.1, 1, 2, 4, 8, 12, 15, 19, 24))
dose <- factor(rep(c(0.1, 0, 0, 0.9, 0, 44, 6, 0.1, 44, 0.9, 0, 6), 11))
```

`radfit` *Rank – Abundance or Dominance / Diversity Models*

Description

Functions construct rank – abundance or dominance / diversity or Whittaker plots and fit broken-stick, pre-emption, log-Normal, Zipf and Zipf-Mandelbrot models of species abundance.

Usage

```
## S3 method for class 'data.frame':
radfit(df, ...)
## S3 method for class 'radfit.frame':
plot(x, order.by, BIC = FALSE, model, legend = TRUE,
     as.table = TRUE, ...)
## Default S3 method:
radfit(x, ...)
## S3 method for class 'radfit':
plot(x, BIC = FALSE, legend = TRUE, ...)
rad.null(x, family=poisson, ...)
rad.preempt(x, family = poisson, ...)
rad.lognormal(x, family = poisson, ...)
rad.zipf(x, family = poisson, ...)
rad.zipfbrot(x, family = poisson, ...)
## S3 method for class 'radline':
plot(x, xlab = "Rank", ylab = "Abundance", type = "b", ...)
## S3 method for class 'radline':
lines(x, ...)
## S3 method for class 'radline':
points(x, ...)
as.rad(x)
## S3 method for class 'rad':
plot(x, xlab = "Rank", ylab = "Abundance", ...)
```

Arguments

`df`	Data frame where sites are rows and species are columns.
`x`	A vector giving species abundances in a site, or an object to be plotted.
`order.by`	A vector used for ordering sites in plots.
`BIC`	Use Bayesian Information Criterion, BIC, instead of Akaike's AIC. The penalty for a parameter is $k = \log(S)$ where S is the number of species, whereas AIC uses $k = 2$.
`model`	Show only the specified model. If missing, AIC is used to select the model. The model names (which can be abbreviated) are `Preemption`, `Lognormal`, `Veiled.LN`, `Zipf`, `Mandelbrot`.
`legend`	Add legend of line colours.
`as.table`	Arrange panels starting from upper left corner (passed to `xyplot`).
`family`	Error distribution (passed to `glm`). All alternatives accepting `link = "log"` in `family` can be used, although not all make sense.
`xlab,ylab`	Labels for `x` and `y` axes.
`type`	Type of the plot, `"b"` for plotting both observed points and fitted lines, `"p"` for only points, `"l"` for only fitted lines, and `"n"` for only setting the frame.
`...`	Other parameters to functions.

Details

Rank – Abundance Dominance (RAD) or Dominance/Diversity plots (Whittaker 1965) display logarithmic species abundances against species rank order. These plots are supposed to be effective in analysing types of abundance distributions in communities. These functions fit some of the most popular models mainly following Wilson (1991). Function `as.rad` constructs observed RAD data. Functions `rad.XXXX` (where `XXXX` is a name) fit the individual models, and function `radfit` fits all models. The argument of the function `radfit` can be either a vector for a single community or a data frame where each row represents a distinct community. All these functions have their own `plot` functions. When the argument is a data frame, `plot` uses `Lattice` graphics, and other functions use ordinary graphics. The ordinary graphics functions return invisibly an `ordiplot` object for observed points, and function `identify.ordiplot` can be used to label selected species. The most complete control of graphics can be achieved with `rad.XXXX` methods which have `points` and `lines` functions to add observed values and fitted models into existing graphs.

Function `rad.null` fits a brokenstick model where the expected abundance of species at rank r is $a_r = (J/S)\sum_{x=r}^{S}(1/x)$ (Pielou 1975), where J is the total number of individuals (site total) and S is the total number of species in the community. This gives a Null model where the individuals are randomly distributed among observed species, and there are no fitted parameters. Function `rad.preempt` fits the niche preemption model, a.k.a. geometric series or Motomura model, where the expected abundance a of species at rank r is $a_r = J\alpha(1-\alpha)^{r-1}$. The only estimated parameter is the preemption coefficient α which gives the decay rate of abundance per rank. The niche preemption model is a straight line in a RAD plot. Function `rad.lognormal` fits a log-Normal model which assumes that the logarithmic abundances are distributed Normally, or $a_r = \exp(\log\mu + \log\sigma N)$, where N is a Normal deviate. Function `rad.zipf` fits the Zipf model $a_r = Jp_1 r^\gamma$ where p_1 is the fitted proportion of the most abundant species, and γ is a decay coefficient. The Zipf – Mandelbrot model (`rad.zipfbrot`) adds one parameter: $a_r = Jc(r+\beta)^\gamma$

after which p_1 of the Zipf model changes into a meaningless scaling constant c. There are grand narratives about ecological mechanisms behind each model (Wilson 1991), but several alternative and contrasting mechanisms can produce similar models and a good fit does not imply a specific mechanism.

Log-Normal and Zipf models are generalized linear models (`glm`) with logarithmic link function. Zipf-Mandelbrot adds one nonlinear parameter to the Zipf model, and is fitted using `nlm` for the nonlinear parameter and estimating other parameters and log-Likelihood with `glm`. Pre-emption model is fitted as purely nonlinear model. There are no estimated parameters in the Null model. The default `family` is `poisson` which is appropriate only for genuine counts (integers), but other families that accept `link = "log"` can be used. Family `Gamma` may be appropriate for abundance data, such as cover. The "best" model is selected by `AIC`. Therefore "quasi" families such as `quasipoisson` cannot be used: they do not have `AIC` nor log-Likelihood needed in non-linear models.

Value

Function `rad.XXXX` will return an object of class `radline`, which is constructed to resemble results of `glm` and has many (but not all) of its components, even when only `nlm` was used in fitting. At least the following `glm` methods can be applied to the result: `fitted`, `residuals.glm` with alternatives `"deviance"` (default), `"pearson"`, `"response"`, function `coef`, `AIC`, `extractAIC`, and `deviance`. Function `radfit` applied to a vector will return an object of class `radfit` with item `y` for the constructed RAD, item `family` for the error distribution, and item `models` containing each `radline` object as an item. In addition, there are special `AIC`, `coef` and `fitted` implementations for `radfit` results. When applied to a data frame `radfit` will return an object of class `radfit.frame` which is a list of `radfit` objects; function `summary` can be used to display the results for individual `radfit` objects. The functions are still preliminary, and the items in the `radline` objects may change.

Note

The RAD models are usually fitted for proportions instead of original abundances. However, nothing in these models seems to require division of abundances by site totals, and original observations are used in these functions. If you wish to use proportions, you must standardize your data by site totals, e.g. with `decostand` and use appropriate `family` such as `Gamma`.

The lognormal model is fitted in a standard way, but I do think this is not quite correct – at least it is not equivalent to fitting Normal density to log abundances like originally suggested (Preston 1948).

Some models may fail. In particular, estimation of the Zipf-Mandelbrot model is difficult. If the fitting fails, `NA` is returned.

Wilson (1991) defined preemption model as $a_r = Jp_1(1-\alpha)^{r-1}$, where p_1 is the fitted proportion of the first species. However, parameter p_1 is completely defined by α since the fitted proportions must add to one, and therefore I handle preemption as a one-parameter model.

Veiled log-Normal model was included in earlier releases of this function, but it was removed because it was flawed: an implicit veil line also appears in the ordinary log-Normal. The latest release version with `rad.veil` was `1.6-10`.

Author(s)

Jari Oksanen

References

Pielou, E.C. (1975) *Ecological Diversity*. Wiley & Sons.

Preston, F.W. (1948) The commonness and rarity of species. *Ecology* 29, 254–283.

Whittaker, R. H. (1965) Dominance and diversity in plant communities. *Science* 147, 250–260.

Wilson, J. B. (1991) Methods for fitting dominance/diversity curves. *Journal of Vegetation Science* 2, 35–46.

See Also

`fisherfit` and `prestonfit`. An alternative approach is to use `qqnorm` or `qqplot` with any distribution. For controlling graphics: `Lattice`, `xyplot`, `lset`.

Examples

```
data(BCI)
mod <- rad.lognormal(BCI[1,])
mod
plot(mod)
mod <- radfit(BCI[1,])
plot(mod)
# Take a subset of BCI to save time and nerves
mod <- radfit(BCI[2:5,])
mod
plot(mod, pch=".")
```

`rankindex` *Compares Dissimilarity Indices for Gradient Detection*

Description

Rank correlations between dissimilarity indices and gradient separation.

Usage

```
rankindex(grad, veg, indices = c("euc", "man", "gow", "bra", "kul"),
          stepacross = FALSE, method = "spearman", ...)
```

Arguments

`grad`	The gradient variable or matrix.
`veg`	The community data matrix.
`indices`	Dissimilarity indices compared, partial matches to alternatives in `vegdist`.
`stepacross`	Use `stepacross` to find a shorter path dissimilarity. The dissimilarities for site pairs with no shared species are set `NA` using `no.shared` so that indices with no fixed upper limit can also be analysed.
`method`	Correlation method used.
`...`	Other parameters to `stepacross`.

Details

A good dissimilarity index for multidimensional scaling should have a high rank-order similarity with gradient separation. The function compares most indices in `vegdist` against gradient separation using rank correlation coefficients in `cor.test`. The gradient separation between each point is assessed as Euclidean distance for continuous variables, and as Gower metric for mixed data using function `daisy` when `grad` has factors.

Value

Returns a named vector of rank correlations.

Note

There are several problems in using rank correlation coefficients. Typically there are very many ties when $n(n-1)/2$ gradient separation values are derived from just n observations. Due to floating point arithmetics, many tied values differ by machine epsilon and are arbitrarily ranked differently by `rank` used in `cor.test`. Two indices which are identical with certain transformation or standardization may differ slightly (magnitude 10^{-15}) and this may lead into third or fourth decimal instability in rank correlations. Small differences in rank correlations should not be taken too seriously. Probably this method should be replaced with a sounder method, but I do not yet know which... You may experiment with `mantel`, `anosim` or even `protest`.

Earlier version of this function used `method = "kendall"`, but that is far too slow in large data sets.

Author(s)

Jari Oksanen

References

Faith, F.P., Minchin, P.R. and Belbin, L. (1987). Compositional dissimilarity as a robust measure of ecological distance. *Vegetatio* 69, 57-68.

See Also

`vegdist`, `stepacross`, `no.shared`, `isoMDS`, `cor`, `Machine`, and for alternatives `anosim`, `mantel` and `protest`.

Examples

```
data(varespec)
data(varechem)
## The next scales all environmental variables to unit variance.
## Some would use PCA transformation.
rankindex(scale(varechem), varespec)
rankindex(scale(varechem), wisconsin(varespec))
```

`read.cep` *Reads a CEP (Canoco) data file*

Description

`read.cep` reads a file formatted by relaxed strict CEP format used by `Canoco` software, among others.

Usage

```
read.cep(file, maxdata=10000, positive=TRUE, trace=FALSE, force=FALSE)
```

Arguments

`file`	File name (character variable).
`maxdata`	Maximum number of non-zero entries.
`positive`	Only positive entries, like in community data.
`trace`	Work verbosely.
`force`	Run function, even if R refuses first.

Details

Cornell Ecology Programs (CEP) introduced several data formats designed for punched cards. One of these was the 'condensed strict' format which was adopted by popular software `DECORANA` and `TWINSPAN`. Later, Cajo ter Braak wrote `Canoco` based on `DECORANA`, where he adopted the format, but relaxed it somewhat (that's why I call it a 'relaxed strict' format). Further, he introduced a more ordinary 'free' format, and allowed the use of classical Fortran style 'open' format with fixed field widths. This function should be able to deal with all these `Canoco` formats, whereas it cannot read many of the traditional CEP alternatives.

All variants of CEP formats have:

- Two or three title cards, most importantly specifying the format (or word `FREE`) and the number of items per record (number of species and sites for `FREE` format).
- Data in one of three accepted formats:
  1. Condensed format: First number on the line is the site identifier, and it is followed by pairs ('couplets') of numbers identifying the species and its abundance (an integer and a floating point number).
  2. Open Fortran format, where the first number on the line must be the site number, followed by abundance values in fields of fixed widths. Empty fields are interpreted as zeros.
  3. 'Free' format, where the numbers are interpreted as abundance values. These numbers must be separated by blank space, and zeros must be written as zeros.
- Species and site names, given in Fortran format `(10A8)`: Ten names per line, eight columns for each.

With option `positive = TRUE` the function removes all lines and columns with zero or negative marginal sums. In community data with only positive entries, this removes empty sites and species. If data entries can be negative, this ruins data, and such data sets should be read in with option `positive = FALSE`.

Value

Returns a data frame, where columns are species and rows are sites. Column and row names are taken from the CEP file, and changed into unique R names by `make.names` after stripping the blanks.

Note

The function relies on smooth linking of Fortran file IO in R session. This is not guaranteed to work, and therefore the function may not work in *your* system, but it can crash the R session. Therefore the default is that the function does not run. If you still want to try:

1. Save your session
2. Run `read.cep()` with switch `force=TRUE`

If you transfer files between operating systems or platforms, you should always check that your file is formatted to your current platform. For instance, if you transfer files from Windows to Linux, you should change the files to `unix` format, or your session may crash when Fortran program tries to read the invisible characters that Windows uses at the end of each line.

If you compiled `vegan` using `gfortran`, the input is probably corrupted. You either should compile `vegan` with other FORTRAN compilers or not to use `read.cep`. The problems still persist in `gfortran 4.01`.

Author(s)

Jari Oksanen

References

Ter Braak, C.J.F. (1984–): CANOCO – a FORTRAN program for *cano*nical *c*ommunity *o*rdination by [partial] [detrended] [canonical] correspondence analysis, principal components analysis and redundancy analysis. *TNO Inst. of Applied Computer Sci., Stat. Dept. Wageningen, The Netherlands*.

Examples

```
## Provided that you have the file `dune.spe'
## Not run:
theclassic <- read.cep("dune.spe", force=T)
## End(Not run)
```

renyi — *Renyi and Hill Diversities and Corresponding Accumulation Curves*

Description

Function `renyi` find Rényi diversities with any scale or the corresponding Hill number (Hill 1973). Function `renyiaccum` finds these statistics with accumulating sites.

Usage

```
renyi(x, scales = c(0, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, Inf), hill = FALSE)
## S3 method for class 'renyi':
plot(x, ...)
renyiaccum(x, scales = c(0, 0.5, 1, 2, 4, Inf), permutations = 100,
    raw = FALSE, ...)
## S3 method for class 'renyiaccum':
plot (x, what = c("mean", "Qnt 0.025", "Qnt 0.975"), type = "l",
    ...)
## S3 method for class 'renyiaccum':
persp (x, theta = 220, col = heat.colors(100), zlim, ...)
rgl.renyiaccum(x, rgl.height = 0.2, ...)
```

Arguments

`x`	Community data matrix or plotting object.
`scales`	Scales of Rényi diversity.
`hill`	Calculate Hill numbers.
`permutations`	Number of random permutations in accumulating sites.
`raw`	if `FALSE` then return summary statistics of permutations, and if `TRUE` then returns the individual permutations.
`what`	Items to be plotted.
`type`	Type of plot, where `type = "l"` means lines.
`theta`	Angle defining the viewing direction (azimuthal) in `persp`.
`col`	Colours used for surface. Single colour will be passed on, and vector colours will be selected by the midpoint of a rectangle in `persp`.
`zlim`	Limits of vertical axis.
`rgl.height`	Scaling of vertical axis.
`...`	Other arguments which are passed to `renyi` and to graphical functions.

Details

Common diversity indices are special cases of Rényi diversity

$$H_a = \frac{1}{1-a} \log \sum p_i^a$$

where a is a scale parameter, and Hill (1975) suggested to use so-called "Hill numbers" defined as $N_a = \exp(H_a)$. Some Hill numbers are the number of species with $a = 0$, $\exp(H')$ or the exponent of Shannon diversity with $a = 1$, inverse Simpson with $a = 2$ and $1/\max(p_i)$ with $a = \infty$. According to the theory of diversity ordering, one community can be regarded as more diverse than another only if its Rényi diversities are all higher (Tóthmérész 1995).

The `plot` method for `renyi` uses **lattice** graphics, and displays the diversity values against each scale in separate panel for each site together with minimum, maximum and median values in the complete data.

Function `renyiaccum` is similar to `specaccum` but finds Rényi or Hill diversities at given `scales` for random permutations of accumulated sites. Its `plot` function uses **lattice** function `xyplot` to display the accumulation curves for each value of `scales` in a separate panel. In addition, it has a `persp` method to plot the diversity surface against scale and number and sites. Dynamic graphics with `rgl.renyiaccum` use **rgl** package, and produces similar surface as `persp` with a mesh showing the empirical confidence levels.

Value

Function `renyi` returns a data frame of selected indices. Function `renyiaccum` with argument `raw = FALSE` returns a three-dimensional array, where the first dimension are the accumulated sites, second dimension are the diveristy scales, and third dimension are the summary statistics `mean`, `stdev`, `min`, `max`, `Qnt 0.025` and `Qnt 0.975`. With argument `raw = TRUE` the statistics on the third dimension are replaced with individual permutation results.

Author(s)

Roeland Kindt ⟨r.kindt@cgiar.org⟩ and Jari Oksanen

References

http://www.worldagroforestry.org/treesandmarkets/tree_diversity_analysis.asp

Hill, M.O. (1973). Diversity and evenness: a unifying notation and its consequences. *Ecology* 54, 427–473.

Kindt R, Van Damme P, Simons AJ. 2006. Tree diversity in western Kenya: using profiles to characterise richness and evenness. *Biodiversity and Conservation* 15: 1253-1270.

Tóthmérész, B. (1995). Comparison of different methods for diversity ordering. *Journal of Vegetation Science* 6, 283–290.

See Also

`diversity` for diversity indices, and `specaccum` for ordinaty species accumulation curves, and `xyplot`, `persp` and `rgl` for controlling graphics.

Examples

```
data(BCI)
i <- sample(nrow(BCI), 12)
mod <- renyi(BCI[i,])
plot(mod)
mod <- renyiaccum(BCI[i,])
plot(mod, as.table=TRUE, col = c(1, 2, 2))
persp(mod)
```

---

scores *Get Species or Site Scores from an Ordination*

Description

Function to access either species or site scores for specified axes in some ordination methods.

Usage

```
## Default S3 method:
scores(x, choices, display=c("sites", "species"), ...)
```

Arguments

`x`	An ordination result.
`choices`	Ordination axes. If missing, returns all axes.
`display`	Partial match to access scores for `sites` or `species`.
`...`	Other parameters (unused).

Details

Functions `cca` and `decorana` have specific `scores` function to access their ordination scores. Most standard ordination methods of libraries **mva**, **multiv** and **MASS** do not have a specific `class`, and no specific method can be written for them. However, `scores.default` guesses where some commonly used functions keep their site scores and possible species scores. For site scores, the function seeks items in order `points`, `rproj`, `x`, and `scores`. For species, the seeking order is `cproj`, `rotation`, and `loadings`. If `x` is a matrix, `scores.default` returns the chosen columns of that matrix, ignoring whether species or sites were requested (do not regard this as a bug but as a feature, please). Currently the function seems to work at least for `isoMDS`, `prcomp`, `princomp`, `ca`, `pca`. It may work in other cases or fail mysteriously.

Value

The function returns a matrix of requested scores.

Author(s)

Jari Oksanen

See Also

`scores.cca`, `scores.decorana`. These have somewhat different interface – `scores.cca` in particular – but all work with keywords `display="sites"` and `display="species"` and return a matrix with these.

Examples

```
data(varespec)
vare.pca <- prcomp(varespec)
scores(vare.pca, choices=c(1,2))
```

`screeplot.cca` *Screeplots for Ordination Results and Broken Stick Distributions*

Description

Screeplot methods for plotting variances of ordination axes/components and overlaying broken stick distributions. Also, provides alternative screeplot methods for `princomp` and `prcomp`.

Usage

```
## S3 method for class 'cca':
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = min(10, if (is.null(x$CCA)) x$CA$rank else x$CCA$rank),
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse(substitute(x)),
         ...)

## S3 method for class 'decorana':
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = 4,
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse(substitute(x)),
         ...)

## S3 method for class 'prcomp':
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = min(10, length(x$sdev)),
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse(substitute(x)),
         ...)

## S3 method for class 'princomp':
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
            npcs = min(10, length(x$sdev)),
            ptype = "o", bst.col = "red", bst.lty = "solid",
            xlab = "Component", ylab = "Inertia",
            main = deparse(substitute(x)),
            ...)

bstick(n, ...)

## Default S3 method:
bstick(n, tot.var = 1, ...)

## S3 method for class 'cca':
bstick(n, ...)

## S3 method for class 'prcomp':
bstick(n, ...)

## S3 method for class 'princomp':
bstick(n, ...)

## S3 method for class 'decorana':
bstick(n, ...)
```

Arguments

`x`	an object from which the component variances can be determined.
`bstick`	logical; should the broken stick distribution be drawn?
`npcs`	the number of components to be plotted.
`type`	the type of plot.
`ptype`	if `type == "lines"` or `bstick = TRUE`, a character indicating the type of plotting used for the lines; actually any of the `type`s as in `plot.default`.
`bst.col, bst.lty`	the colour and line type used to draw the broken stick distribution.
`xlab, ylab, main`	graphics parameters.
`n`	an object from which the variances can be extracted or the number of variances (components) in the case of `bstick.default`.
`tot.var`	the total variance to be split.
`...`	arguments passed to other methods.

Details

The functions provide screeplots for most ordination methods in **vegan** and enhanced versions with broken stick for `prcomp` and `princomp`.

Function `bstick` gives the brokenstick values which are ordered random proportions, defined as $p_i = (tot/n)\sum_{x=i}^{n}(1/x)$ (Legendre & Legendre 1998), where tot is the total and n is the number of brokenstick components (cf. radfit). Broken stick has been recommended as a stopping rule in principal component analysis (Jackson 1993): principal components should be retained as long as observed eigenvalues are higher than corresponding random broken stick components.

The bstick function is generic. The default needs the number of components and the total, and specific methods extract this information from ordination results. There also is a bstick method for cca. However, the broken stick model is not strictly valid for correspondence analysis (CA), because eigenvalues of CA are defined to be ≤ 1, whereas brokenstick components have no such restrictions. The brokenstick components are not available for decorana where the sum of eigenvalues (total inertia) is unknown, and the eigenvalues of single axes are not additive in detrended analysis.

Value

Function screeplot draws a plot on the currently active device, and returns invisibly the xy.coords of the points or bars for the eigenvalues.

Function bstick returns a numeric vector of broken stick components.

Note

Function screeplot is generic from R version 2.5.0. In these versions you can use plain screeplot command without suffices cca, prcomp etc.

Author(s)

Gavin L. Simpson

References

Jackson, D. A. (1993). Stopping rules in principal components analysis: a comparison of heuristical and statistical approaches. *Ecology* 74, 2204–2214.

Legendre, P. and Legendre, L. (1998) *Numerical Ecology*. 2nd English ed. Elsevier.

See Also

cca, decorana, princomp and prcomp for the ordination functions, and screeplot for the stock version.

Examples

```
data(varespec)
vare.pca <- rda(varespec, scale = TRUE)
bstick(vare.pca)
screeplot.cca(vare.pca, bstick = TRUE, type = "lines")
```

sipoo *Birds in the Archipelago of Sipoo (Sibbo)*

Description

Land birds on islands covered by coniferous forest in the Sipoo archipelago, southern Finland (land-bridge/ oceanic distinction unclear from source).

Usage

```
data(sipoo)
```

Format

A data frame with 18 sites and 50 species (Simberloff & Martin, 1991, Appendix 3). The species are referred by 4+4 letter abbreviation of their Latin names (but using five letters in two species names to make these unique). The example gives the areas of the studies islands in hectares.

Source

http://www.aics-research.com/nested/

References

Simberloff, D. & Martin, J.-L. (1991). Nestedness of insular avifaunas: simple summary statistics masking complex species patterns. *Ornis Fennica* 68:178–192.

Examples

```
data(sipoo)
## Areas of the islands in hectares
sipoo.area <-  c(1.1, 2.1, 2.2, 3.1, 3.5, 5.8, 6, 6.1, 6.5, 11.4, 13,
14.5, 16.1 ,17.5, 28.7, 40.5, 104.5, 233)
```

spantree *Minimum Spanning Tree*

Description

Function `spantree` finds a minimum spanning tree connecting all points, but disregarding dissimilarities that are at or above the threshold or `NA`.

Usage

```
spantree(dis, toolong = 0)
## S3 method for class 'spantree':
cophenetic(x)
## S3 method for class 'spantree':
plot(x, ord, cex = 0.7, type = "p", labels, dlim,
     FUN = sammon,  ...)
## S3 method for class 'spantree':
lines(x, ord, display="sites", ...)
```

Arguments

`dis`	Dissimilarity data inheriting from class `dist` or a an object, such as a matrix, that can be converted to a dissimilarity matrix. Functions `vegdist` and `dist` are some functions producing suitable dissimilarity data.
`toolong`	Shortest dissimilarity regarded as `NA`. The function uses a fuzz factor, so that dissimilarities close to the limit will be made `NA`, too. If `toolong = 0` (or negative), no dissimmilarity is regarded as too long.
`x`	A `spantree` result object.
`ord`	An ordination configuration, or an ordination result known by `scores`.
`cex`	Character expansion factor.
`type`	Observations are plotted as points with `type="p"` or `type="b"`, or as text with `type="t"`. The tree (lines) will always be plotted.
`labels`	Text used with `type="t"` or row names if this is missing.
`dlim`	A ceiling value used to highest `cophenetic` dissimilarity.
`FUN`	Ordination function to find the configuration from cophenetic dissimilarities.
`display`	Type of `scores` used for `ord`.
`...`	Other parameters passed to functions.

Details

Function `spantree` finds a minimum spanning tree for dissimilarities (there may be several minimum spanning trees, but the function finds only one). Dissimilarities at or above the threshold `toolong` and `NA`s are disregarded, and the spanning tree is found through other dissimilarities. If the data are disconnected, the function will return a disconnected tree (or a forest), and the corresponding link is `NA`. Connected subtrees can be identified using `distconnected`.

Function `cophenetic` finds distances between all points along the tree segments. Function `plot` displays the tree over a supplied ordination configuration, and `lines` adds a spanning tree to an ordination graph. If configuration is not supplied for `plot`, the function ordinates the the cophenetic dissimilarities of the spanning tree and overlays the tree on this result. The default ordination function is `sammon` (package **MASS**), because Sammon scaling emphasizes structure in the neighbourhood of nodes and may be able to beautifully represent the tree (you may need to set `dlim`, and sometimes the results will remain twisted). These ordination methods do not work with disconnected trees, but you must supply the ordination configuration. Function `lines` will overlay the tree in an existing plot.

Function `spantree` uses Prim's method implemented as priority-first search for dense graphs (Sedgewick 1990). Function `cophenetic` uses function `stepacross` with option `path = "extended"`. The `spantree` is very fast, but `cophenetic` is slow in very large data sets.

Value

Function `spantree` returns an object of class `spantree` which is a list with two vectors, each of length $n - 1$. The number of links in a tree is one less the number of observations, and the first item is omitted. The items are

`kid`	The child node of the parent, starting from parent number two. If there is no link from the parent, value will be `NA` and tree is disconnected at the node.
`dist`	Corresponding distance. If `kid = NA`, then `dist = 0`.

Note

In principle, minimum spanning tree is equivalent to single linkage clustering that can be performed using `hclust` or `agnes`. However, these functions combine clusters to each other and the information of the actually connected points (the "single link") cannot be recovered from the result. The graphical output of a single linkage clustering plotted with `ordicluster` will look very different from an equivalent spanning tree plotted with `lines.spantree`.

Author(s)

Jari Oksanen

References

Sedgewick, R. (1990). *Algorithms in C*. Addison Wesley.

See Also

`vegdist` or `dist` for getting dissimilarities, and `hclust` or `agnes` for single linkage clustering.

Examples

```
data(dune)
dis <- vegdist(dune)
tr <- spantree(dis)
## Add tree to a metric scaling
plot(tr, cmdscale(dis))
## Find a configuration to display the tree neatly
plot(tr)
```

`specaccum` *Species Accumulation Curves*

Description

Function `specaccum` finds species accumulation curves or the number of species for a certain number of sampled sites or individuals.

Usage

```
specaccum(comm, method = "exact", permutations = 100,
          conditioned =TRUE, gamma = "Jack.1",  ...)
## S3 method for class 'specaccum':
plot(x, add = FALSE, ci = 2, ci.type = c("bar", "line", "polygon"),
    col = par("fg"), ci.col = col, ci.lty = 1, xlab = "Sites",
    ylab = x$method, ylim, ...)
## S3 method for class 'specaccum':
boxplot(x, add = FALSE, ...)
```

Arguments

`comm`	Community data set.
`method`	Species accumulation method (partial match). Method `"collector"` adds sites in the order they happen to be in the data, `"random"` adds sites in random order, `"exact"` finds the expected (mean) species richness, `"coleman"` finds the expected richness following Coleman et al. 1982, and `"rarefaction"` finds the mean when accumulating individuals instead of sites.
`permutations`	Number of permutations with `method = "random"`.
`conditioned`	Estimation of standard deviation is conditional on the empirical dataset for the exact SAC
`gamma`	Method for estimating the total extrapolated number of species in the survey area by function `specpool`
`x`	A `specaccum` result object
`add`	Add to an existing graph.
`ci`	Multiplier used to get confidence intervals from standard deviation (standard error of the estimate). Value `ci = 0` suppresses drawing confidence intervals.
`ci.type`	Type of confidence intervals in the graph: `"bar"` draws vertical bars, `"line"` draws lines, and `"polygon"` draws a shaded area.
`col`	Colour for drawing lines.
`ci.col`	Colour for drawing lines or filling the `"polygon"`.
`ci.lty`	Line type for confidence intervals or border of the `"polygon"`.
`xlab,ylab`	Labels for `x` and `y` axis.
`ylim`	the y limits of the plot.
`...`	Other parameters to functions.

Details

Species accumulation curves (SAC) are used to compare diversity properties of community data sets using different accumulator functions. The classic method is `"random"` which finds the mean SAC and its standard deviation from random permutations of the data, or subsampling without replacement (Gotelli & Colwell 2001). The `"exact"` method finds the expected SAC using the method that was independently developed by Ugland et al. (2003), Colwell et al. (2004) and Kindt et al. (2006). The unconditional standard deviation for the exact SAC represents a moment-based estimation that is not conditioned on the empirical data set (sd for all samples > 0), unlike the conditional standard deviation that was developed by Jari Oksanen (not published, sd=0 for all samples). The unconditional standard deviation is based on an estimation of the total extrapolated number of species in the survey area (a.k.a. gamma diversity), as estimated by function `specpool`. Method `"coleman"` finds the expected SAC and its standard deviation following Coleman et al. (1982). All these methods are based on sampling sites without replacement. In contrast, the `method = "rarefaction"` finds the expected species richness and its standard deviation by sampling individuals instead of sites. It achieves this by applying function `rarefy` with number of individuals corresponding to average number of individuals per site.

The function has a `plot` method. In addition, `method = "random"` has `summary` and `boxplot` methods.

Value

The function returns an object of class `"specaccum"` with items:

`call`	Function call.
`method`	Accumulator method.
`sites`	Number of sites. For `method = "rarefaction"` this is the number of sites corresponding to a certain number of individuals and generally not an integer, and the average number of individuals is also retunred in item `individuals`.
`richness`	The number of species corresponding to number of sites. With `method = "collector"` this is the observed richness, for other methods the average or expected richness.
`sd`	The standard deviation of SAC (or its standard error). This is `NULL` in `method = "collector"`, and it is estimated from permutations in `method = "random"`, and from analytic equations in other methods.
`perm`	Permutation results with `method = "random"` and `NULL` in other cases. Each column in `perm` holds one permutation.

Note

The SAC with `method = "exact"` was developed by Roeland Kindt, and its standard deviation by Jari Oksanen (both are unpublished). The `method = "coleman"` underestimates the SAC because it does not handle properly sampling without replacement. Further, its standard deviation does not take into account species correlations, and is generally too low.

Author(s)

Roeland Kindt ⟨r.kindt@cgiar.org⟩ and Jari Oksanen.

References

Coleman, B.D, Mares, M.A., Willis, M.R. & Hsieh, Y. (1982). Randomness, area and species richness. *Ecology* 63: 1121–1133.

Colwell, R.K., Mao, C.X. & Chang, J. (2004). Interpolating, extrapolating, and comparing incidence-based species accumulation curves. *Ecology* 85: 2717–2727.

Gotellli, N.J. & Colwell, R.K. (2001). Quantifying biodiversity: procedures and pitfalls in measurement and comparison of species richness. *Ecol. Lett.* 4, 379–391.

Kindt, R. (2003). Exact species richness for sample-based accumulation curves. *Manuscript.*

Kindt R., Van Damme, P. & Simons, A.J. (2006) Patterns of species richness at varying scales in western Kenya: planning for agroecosystem diversification. *Biodiversity and Conservation*, online first: DOI 10.1007/s10531-005-0311-9

Ugland, K.I., Gray, J.S. & Ellingsen, K.E. (2003). The species-accumulation curve and estimation of species richness. *Journal of Animal Ecology* 72: 888–897.

See Also

`rarefy` and `renyiaccum`. Underlying graphical functions are `boxplot`, `matlines`, `segments` and `polygon`.

Examples

```
data(BCI)
sp1 <- specaccum(BCI)
sp2 <- specaccum(BCI, "random")
sp2
summary(sp2)
plot(sp1, ci.type="poly", col="blue", lwd=2, ci.lty=0, ci.col="lightblue")
boxplot(sp2, col="yellow", add=TRUE, pch="+")
```

`specpool` *Extrapolated Species Richness in a Species Pool*

Description

The functions estimate the extrapolated species richness in a species pool, or the number of unobserved species. Function `specpool` is based on incidences in sample sites, and gives a single estimate for a collection of sample sites (matrix). Function `estimateR` is based on abundances (counts) on single sample site.

Usage

```
specpool(x, pool)
specpool2vect(X, index = c("Jack.1","Jack.2", "Chao", "Boot","Species"))
estimateR(x, ...)
```

Arguments

`x`	Data frame or matrix with species data.
`pool`	A vector giving a classification for pooling the sites in the species data. If missing, all sites are pooled together.
`X`	A `specpool` result object.
`index`	The selected index of extrapolated richness.
`...`	Other parameters (not used).

Details

Many species will always remain unseen or undetected in a collection of sample plots. The function uses some popular ways of estimating the number of these unseen species and adding them to the observed species richness (Palmer 1990, Colwell & Coddington 1994).

The incidence-based estimates in `specpool` use the frequencies of species in a collection of sites. In the following, S_P is the extrapolated richness in a pool, S_0 is the observed number of species in the collection, a_1 and a_2 are the number of species occurring only in one or only in two sites in the collection, p_i is the frequency of species i, and N is the number of sites in the collection. The variants of extrapolated richness in `specpool` are:

Chao	$S_P = S_0 + a1^2/(2 * a2)$
First order jackknife	$S_P = S_0 + a_1 \frac{N-1}{N}$
Second order jackknife	$S_P = S_0 + a_1 \frac{2N-3}{N} - a_2 \frac{(N-2)^2}{N(N-1)}$
Bootstrap	$S_P = S_0 + \sum_{i=1}^{S_0} (1 - p_i)^N$

The abundance-based estimates in `estimateR` use counts (frequencies) of species in a single site. If called for a matrix or data frame, the function will give separate estimates for each site. The two variants of extrapolated richness in `estimateR` are Chao (unbiased variant) and ACE. In the Chao estimate a_i refers to number of species with abundance i instead of incidence:

Chao	$S_P = S_0 + \frac{a_1(a_1-1)}{2(a_2+1)}$
ACE	$S_P = S_{abund} + \frac{S_{rare}}{C_{ace}} + \frac{a_1}{C_{ace}} \gamma^2_{ace}$
where	$C_{ace} = 1 - \frac{a_1}{N_{rare}}$
	$\gamma^2_{ace} = \max \left[\frac{S_{rare} \sum_{i=1}^{10} i(i-1)a_i}{C_{ace} N_{rare}(N_{rare}-1)} - 1, 0 \right]$

Here a_i refers to number of species with abundance i and S_{rare} is the number of rare species, S_{abund} is the number of abundant species, with an arbitrary threshold of abundance 10 for rare species, and N_{rare} is the number of individuals in rare species.

Functions estimate the the standard errors of the estimates. These only concern the number of added species, and assume that there is no variance in the observed richness. The equations of standard errors are too complicated to be reproduced in this help page, but they can be studied in the R source code of the function. The standard error are based on the following sources: Chao (1987) for the Chao estimate and Smith and van Belle (1984) for the first-order Jackknife and the bootstrap (second-order jackknife is still missing). The variance estimator of S_{ace} was developed by Bob O'Hara (unpublished).

Value

Function `specpool` returns a data frame with entries for observed richness and each of the indices for each class in `pool` vector. The utility function `specpool2vect` maps the pooled values into a vector giving the value of selected `index` for each original site. Function `estimateR` returns the estimates and their standard errors for each site.

Note

The functions are based on assumption that there is a species pool: The community is closed so that there is a fixed pool size S_P. Such cases may exist, although I have not seen them yet. All indices are biased for open communities.

See `http://viceroy.eeb.uconn.edu/EstimateS` for a more complete (and positive) discussion and alternative software for some platforms.

Author(s)

Bob O'Hara (`estimateR`) and Jari Oksanen (`specpool`).

References

Chao, A. (1987). Estimating the population size for capture-recapture data with unequal catchability. *Biometrics* 43, 783–791.

Colwell, R.K. & Coddington, J.A. (1994). Estimating terrestrial biodiversity through extrapolation. *Phil. Trans. Roy. Soc. London* B 345, 101–118.

Palmer, M.W. (1990). The estimation of species richness by extrapolation. *Ecology* 71, 1195–1198.

Smith, E.P & van Belle, G. (1984). Nonparametric estimation of species richness. *Biometrics* 40, 119–129.

See Also

`veiledspec`, `diversity`, `beals`.

Examples

```
data(dune)
data(dune.env)
attach(dune.env)
pool <- specpool(dune, Management)
pool
op <- par(mfrow=c(1,2))
boxplot(specnumber(dune) ~ Management, col="hotpink", border="cyan3",
 notch=TRUE)
boxplot(specnumber(dune)/specpool2vect(pool) ~ Management, col="hotpink",
 border="cyan3", notch=TRUE)
par(op)
data(BCI)
estimateR(BCI[1:5,])
```

`stepacross` *Stepacross as Flexible Shortest Paths or Extended Dissimilarities*

Description

Function `stepacross` tries to replace dissimilarities with shortest paths stepping across intermediate sites while regarding dissimilarities above a threshold as missing data (`NA`). With `path = "shortest"` this is the flexible shortest path (Williamson 1978, Bradfield & Kenkel 1987), and with `path = "extended"` an approximation known as extended dissimilarities (De'ath 1999). The use of `stepacross` should improve the ordination with high beta diversity, when there are many sites with no species in common.

Usage

```
stepacross(dis, path = "shortest", toolong = 1, trace = TRUE, ...)
```

Arguments

`dis`	Dissimilarity data inheriting from class `dist` or a an object, such as a matrix, that can be converted to a dissimilarity matrix. Functions `vegdist` and `dist` are some functions producing suitable dissimilarity data.
`path`	The method of stepping across (partial match) Alternative `"shortest"` finds the shortest paths, and `"extended"` their approximation known as extended dissimilarities.
`toolong`	Shortest dissimilarity regarded as `NA`. The function uses a fuzz factor, so that dissimilarities close to the limit will be made `NA`, too.
`trace`	Trace the calculations.
`...`	Other parameters (ignored).

Details

Williamson (1978) suggested using flexible shortest paths to estimate dissimilarities between sites which have nothing in common, or no shared species. With `path = "shortest"` function `stepacross` replaces dissimilarities that are `toolong` or longer with `NA`, and tries to find shortest paths between all sites using remaining dissimilarities. Several dissimilarity indices are semimetric which means that they do not obey the triangle inequality $d_{ij} \leq d_{ik} + d_{kj}$, and shortest path algorithm can replace these dissimilarities as well, even when they are shorter than `toolong`.

De'ath (1999) suggested a simplified method known as extended dissimilarities, which are calculated with `path = "extended"`. In this method, dissimilarities that are `toolong` or longer are first made `NA`, and then the function tries to replace these `NA` dissimilarities with a path through single stepping stone points. If not all `NA` could be replaced with one pass, the function will make new passes with updated dissimilarities as long as all `NA` are replaced with extended dissimilarities. This mean that in the second and further passes, the remaining `NA` dissimilarities are allowed to have more than one stepping stone site, but previously replaced dissimilarities are not updated. Further, the function does not consider dissimilarities shorter than `toolong`, although some of these could be replaced with a shorter path in semi-metric indices, and used as a part of other paths. In optimal cases, the extended dissimilarities are equal to shortest paths, but they may be longer.

As an alternative to defining too long dissimilarities with parameter `toolong`, the input dissimilarities can contain `NA`s. If `toolong` is zero or negative, the function does not make any dissimilarities into `NA`. If there are no `NA`s in the input and `toolong = 0`, `path = "shortest"` will find shorter paths for semi-metric indices, and `path = "extended"` will do nothing. Function `no.shared` can be used to set dissimilarities to `NA`.

If the data are disconnected or there is no path between all points, the result will contain `NA`s and a warning is issued. Several methods cannot handle `NA` dissimilarities, and this warning should be taken seriously. Function `distconnected` can be used to find connected groups and remove rare outlier observations or groups of observations.

Alternative `path = "shortest"` uses Dijkstra's method for finding flexible shortest paths, implemented as priority-first search for dense graphs (Sedgewick 1990). Alternative `path = "extended"` follows De'ath (1999), but implementation is simpler than in his code.

Value

Function returns an object of class `dist` with extended dissimilarities (see functions `vegdist` and `dist`). The value of `path` is appended to the `method` attribute.

Note

The function changes the original dissimilarities, and not all like this. It may be best to use the function only when you really *must*: extremely high beta diversity where a large proportion of dissimilarities are at their upper limit (no species in common).

Semi-metric indices vary in their degree of violating the triangle inequality. Morisita and Horn–Morisita indices of `vegdist` may be very strongly semi-metric, and shortest paths can change these indices very much. Mountford index violates basic rules of dissimilarities: non-identical sites have zero dissimilarity if species composition of the poorer site is a subset of the richer. With Mountford index, you can find three sites i, j, k so that $d_{ik} = 0$ and $d_{jk} = 0$, but $d_{ij} > 0$. The results of `stepacross` on Mountford index can be very weird. If `stepacross` is needed, it is best to try to use it with more metric indices only.

Author(s)

Jari Oksanen

References

Bradfield, G.E. & Kenkel, N.C. (1987). Nonlinear ordination using flexible shortest path adjustment of ecological distances. *Ecology* 68, 750–753.

De'ath, G. (1999). Extended dissimilarity: a method of robust estimation of ecological distances from high beta diversity data. *Plant Ecol.* 144, 191–199.

Sedgewick, R. (1990). *Algorithms in C*. Addison Wesley.

Williamson, M.H. (1978). The ordination of incidence data. *J. Ecol.* 66, 911-920.

See Also

Function distconnected can find connected groups in disconnected data, and function no.shared can be used to set dissimilarities as NA. See swan for an alternative approach. Function stepacross is an essential component in isomap and cophenetic.spantree.

Examples

```
# There are no data sets with high beta diversity in vegan, but this
# should give an idea.
data(dune)
dis <- vegdist(dune)
edis <- stepacross(dis)
plot(edis, dis, xlab = "Shortest path", ylab = "Original")
## Manhattan distance have no fixed upper limit.
dis <- vegdist(dune, "manhattan")
is.na(dis) <- no.shared(dune)
dis <- stepacross(dis, toolong=0)
```

---

taxondive *Indices of Taxonomic Diversity and Distinctness*

Description

Function finds indices of taxonomic diversity and distinctiness, which are averaged taxonomic distances among species or individuals in the community (Clarke & Warwick 1998, 2001)

Usage

```
taxondive(comm, dis, match.force = FALSE)
taxa2dist(x, varstep = FALSE, check = TRUE, labels)
```

Arguments

comm	Community data.
dis	Taxonomic distances among taxa in comm. This should be a dist object or a symmetric square matrix.
match.force	Force matching of column names in comm and labels in dis. If FALSE, matching only happens when dimensions differ, and in that case the species must be in identical order in both.
x	Classification table with a row for each species or other basic taxon, and columns for identifiers of its classication at higher levels.
varstep	Vary step lengths between successive levels relative to proportional loss of the number of distinct classes.

`check`	If `TRUE`, remove all redundant levels which are different for all rows or constant for all rows and regard each row as a different basal taxon (species). If `FALSE` all levels are retained and basal taxa (species) also must be coded as variables (columns). You will get a warning if species are not coded, but you can ignore this if that was your intention.
`labels`	The `labels` attribute of taxonomic distances. Row names will be used if this is not given. Species will be matched by these labels in `comm` and `dis` in `taxondive` if these have different dimensions.

Details

Clarke & Warwick (1998, 2001) suggested several alternative indices of taxonomic diversity or distinctness. Two basic indices are called taxonomic diversity (Δ) and distinctness (Δ^*):

$$\Delta = (\sum \sum_{i<j} \omega_{ij} x_i x_j)/(n(n-1)/2)$$
$$\Delta^* = (\sum \sum_{i<j} \omega_{ij} x_i x_j)/(\sum \sum_{i<j} x_i x_j)$$

The equations give the index value for a single site, and summation goes over species i and j. Here ω are taxonomic distances among taxa, and x are species abundances, and n is the total abundance for a site. With presence/absence data both indices reduce to the same index Δ^+, and for this index Clarke & Warwick (1998) also have an estimate of its standard deviation. Clarke & Warwick (2001) presented two new indices: $s\Delta^+$ is the product of species richness and Δ^+, and index of variation in taxonomic distinctness (Λ^+) defined as

$$\Lambda^+ = (\sum \sum_{i<j} \omega_{ij}^2)/(n(n-1)/2) - (\Delta^+)^2$$

The `dis` argument must be species dissimilarities. These must be similar to dissimilarities produced by `dist`. It is customary to have integer steps of taxonomic hierarchies, but other kind of dissimilarities can be used, such as those from phylogenetic trees or genetic differences. Further, the `dis` need not be taxonomic, but other species classifications can be used.

Function `taxa2dist` can produce a suitable `dist` object from a classification table. Each species (or basic taxon) corresponds to a row of the classification table, and columns give the classification at different levels. With `varstep = FALSE` the successive levels will be separated by equal steps, and with `varstep = TRUE` the step length is relative to the proportional decrease in the number of classes (Clarke & Warwick 1999). With `check = TRUE`, the function removes classes which are distinct for all species or which combine all species into one class, and assumes that each row presents a distinct basic taxon. The function scales the distances so that longesth path length between taxa is 100 (not necessarily when `check = FALSE`).

Function `plot.taxondive` plots Δ^+ against Number of species, together with expectation and its approximate 2*sd limits. Function `summary.taxondive` finds the z values and their significances from Normal distribution for Δ^+.

Value

Function returns an object of class `taxondive` with following items:

`Species`	Number of species for each site.

`D, Dstar, Dplus, SDplus, Lambda`
: Δ, Δ^*, Δ^+, $s\Delta^+$ and Λ^+ for each site.

`sd.Dplus`
: Standard deviation of Δ^+.

`ED, EDstar, EDplus`
: Expected values of corresponding statistics.

Function `taxa2dist` returns an object of class `"dist"`, with an attribute `"steps"` for the step lengths between successive levels.

Note

The function is still preliminary and may change. The scaling of taxonomic dissimilarities influences the results. If you multiply taxonomic distances (or step lengths) by a constant, the values of all Deltas will be multiplied with the same constant, and the value of Λ^+ by the square of the constant.

Author(s)

Jari Oksanen

References

Clarke, K.R & Warwick, R.M. (1998) A taxonomic distinctness index and its statistical properties. *Journal of Applied Ecology* 35, 523–531.

Clarke, K.R. & Warwick, R.M. (1999) The taxonomic distinctness measure of biodiversity: weighting of step lengths between hierarchical levels. *Marine Ecology Progress Series* 184: 21–29.

Clarke, K.R. & Warwick, R.M. (2001) A further biodiversity index applicable to species lists: variation in taxonomic distinctness. *Marine Ecology Progress Series* 216, 265–278.

See Also

`diversity`.

Examples

```
## Preliminary: needs better data and some support functions
data(dune)
data(dune.taxon)
# Taxonomic distances from a classification table with variable step lengths.
taxdis <- taxa2dist(dune.taxon, varstep=TRUE)
plot(hclust(taxdis), hang = -1)
# Indices
mod <- taxondive(dune, taxdis)
mod
summary(mod)
plot(mod)
```

treedive	*Functional Diversity estimated from a Species Dendrogram*

Description

Functional diversity is defined as the total branch length in a trait dendrogram connecting all species, but excluding the unnecessary root segments of the tree (Petchey and Gaston 2006).

Usage

```
treedive(comm, tree, match.force = FALSE)
treeheight(tree)
```

Arguments

`comm`	Community data frame or matrix.
`tree`	A dendrogram which for `treedive` must be for species (columns).
`match.force`	Force matching of column names in `comm` and labels in `tree`. If `FALSE`, matching only happens when dimensions differ, and in that case the species must be in identical order in both.

Details

Function `treeheight` finds the sum of lengths of connecting segments in a dendrogram produced by `hclust`, or other dendrogram that can be coerced to a correct type using `as.hclust`. When applied to a clustering of species traits, this is a measure of functional diversity (Petchey and Gaston 2002, 2006).

Function `treedive` finds the `treeheight` for each site (row) of a community matrix. The function uses a subset of dendrogram for those species that occur in each site, and excludes the tree root if that is not needed to connect the species (Petchey and Gaston 2006). The subset of the dendrogram is found by first calculating `cophenetic` distances from the input dendrogram, then reconstructing the dendrogram for the subset of the cophenetic distance matrix for species occuring in each site.

The functions need a dendrogram of species traits as an input. If species traits contain `factor` or `ordered` factor variables, it is recommended to use Gower distances for mixed data (function `daisy` in package **cluster**), and usually the recommended clustering method is UPGMA (`method = "average"` in function `hclust`) (Podani and Schmera 2006).

It is possible to analyse the non-randomness of functional diversity using `oecosimu`. This provided specifying an adequate Null model, and the results will change with this choice.

Value

A vector of diversity values or a single tree height.

Author(s)

Jari Oksanen

References

Petchey, O.L. and Gaston, K.J. 2002. Funcional diversity (FD), species richness and community composition. *Ecology Letters* 5, 402–411.

Petchey, O.L. and Gaston, K.J. 2006. Functional diversity: back to basics and looking forward. *Ecology Letters* 9, 741–758.

Podani J. and Schmera, D. 2006. On dendrogram-based methods of functional diversity. *Oikos* 115, 179–185.

See Also

`taxondive` is something very similar from another world.

Examples

```
## There is no data set on species properties yet, and therefore
## the example uses taxonomy
data(dune)
data(dune.taxon)
d <- taxa2dist(dune.taxon, varstep=TRUE)
cl <- hclust(d, "aver")
treedive(dune, cl)
## Significance test using Null model communities.
## The current choice fixes both species and site totals.
oecosimu(dune, treedive, "quasiswap", tree = cl)
```

`tsallis` *Tsallis Diversity and Corresponding Accumulation Curves*

Description

Function `tsallis` find Tsallis diversities with any scale or the corresponding evenness measures. Function `tsallisaccum` finds these statistics with accumulating sites.

Usage

```
tsallis(x, scales = seq(0, 2, 0.2), norm = FALSE)
tsallisaccum(x, scales = seq(0, 2, 0.2), permutations = 100, raw = FALSE, ...)
## S3 method for class 'tsallisaccum':
persp(x, theta = 220, phi = 15, col = heat.colors(100), zlim, ...)
```

Arguments

`x`	Community data matrix or plotting object.
`scales`	Scales of Tsallis diversity.
`norm`	Logical, if `TRUE` diversity values are normalized by their maximum (diversity value at equiprobability conditions).

`permutations`	Number of random permutations in accumulating sites.
`raw`	If `FALSE` then return summary statistics of permutations, and if TRUE then returns the individual permutations.
`theta, phi`	angles defining the viewing direction. `theta` gives the azimuthal direction and `phi` the colatitude.
`col`	Colours used for surface.
`zlim`	Limits of vertical axis.
`...`	Other arguments which are passed to `tsallis` and to graphical functions.

Details

The Tsallis diversity (also equivalent to Patil and Taillie diversity) is a one-parametric generalised entropy function, defined as:

$$H_q = \frac{1}{q-1}(1 - \sum_{i=1}^{S} p_i^q)$$

where q is a scale parameter, S the number of species in the sample (Tsallis 1988, Tothmeresz 1995). This diversity is concave for all $q > 0$, but non-additive (Keylock 2005). For $q = 0$ it gives the number of species minus one, as q tends to 1 this gives Shannon diversity, for $q = 2$ this gives the Simpson index (see function `diversity`).

When `norm = TRUE`, `tsallis` gives values normalized by the maximum:

$$H_q(max) = \frac{S^{1-q} - 1}{1 - q}$$

where S is the number of species. As q tends to 1, maximum is defined as $ln(S)$.

Details on plotting methods and accumulating values can be found on the help pages of the functions `renyi` and `renyiaccum`.

Value

Function `tsallis` returns a data frame of selected indices. Function `tsallisaccum` with argument `raw = FALSE` returns a three-dimensional array, where the first dimension are the accumulated sites, second dimension are the diveristy scales, and third dimension are the summary statistics `mean`, `stdev`, `min`, `max`, `Qnt 0.025` and `Qnt 0.975`. With argument `raw = TRUE` the statistics on the third dimension are replaced with individual permutation results.

Author(s)

Péter Sólymos, ⟨solymos@ualberta.ca⟩, based on the code of Roeland Kindt and Jari Oksanen written for `renyi`

References

Tsallis, C. (1988). Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phis.* 52, 479–487.

Tothmeresz, B. (1995). Comparison of different methods for diversity ordering. *Journal of Vegetation Science* 6, 283–290.

Patil, GP and Taillie, C. (1982). Diversity as a concep and its measurement. *J. Am. Stat. Ass.* 77, 548–567.

Keylock, CJ (2005). Simpson diversity and the Shannon-Wiener index as special cases of a generalized entropy. *Oikos* 109, 203–207.

See Also

Plotting methods and accumulation routins are based on functions `renyi` and `renyiaccum`. An object of class 'tsallisaccum' can be used with function `rgl.renyiaccum` as well. See also settings for `persp`.

Examples

```
data(BCI)
i <- sample(nrow(BCI), 12)
x1 <- tsallis(BCI[i,])
x1
diversity(BCI[i,],"simpson") == x1[["2"]]
plot(x1)
x2 <- tsallis(BCI[i,],norm=TRUE)
x2
plot(x2)
mod1 <- tsallisaccum(BCI[i,])
plot(mod1, as.table=TRUE, col = c(1, 2, 2))
persp(mod1)
mod2 <- tsallisaccum(BCI[i,], norm=TRUE)
persp(mod2,theta=100,phi=30)
```

`varespec` *Vegetation and environment in lichen pastures*

Description

The `varespec` data frame has 24 rows and 44 columns. Columns are estimated cover values of 44 species. The variable names are formed from the scientific names, and are self explanatory for anybody familiar with the vegetation type. The `varechem` data frame has 24 rows and 14 columns, giving the soil characteristics of the very same sites as in the `varespec` data frame. The chemical measurements have obvious names. `Baresoil` gives the estimated cover of bare soil, `Humpdepth` the thickness of the humus layer.

Usage

```
        data(varechem)
        data(varespec)
```

References

Väre, H., Ohtonen, R. and Oksanen, J. (1995) Effects of reindeer grazing on understorey vegetation in dry Pinus sylvestris forests. *Journal of Vegetation Science* 6, 523–530.

Examples

```
data(varespec)
data(varechem)
```

`varpart` *Partition the Variation of Community Matrix by 2, 3, or 4 Explanatory Matrices*

Description

The function partitions the variation of response table Y with respect to two, three, or four explanatory tables, using redundancy analysis ordination (RDA). If Y contains a single vector, partitioning is by partial regression. Collinear variables in the explanatory tables do NOT have to be removed prior to partitioning.

Usage

```
varpart(Y, X, ..., data, transfo, scale = FALSE)
showvarparts(parts, labels, ...)
## S3 method for class 'varpart234':
plot(x, cutoff = 0, digits = 1, ...)
```

Arguments

`Y`	Data frame or matrix containing the response data table. In community ecology, that table is often a site-by-species table.
`X`	Two to four explanatory models, variables or tables. These can be defined in three alternative ways: (1) one-sided model formulae beginning with ~ and then defining the model, (2) name of a single numeric variable, or (3) name of data frame or matrix with numeric variables. The model formulae can have factors, interaction terms and transformations of variables. The names of the variables in the model formula are found in data frame given in `data` argument, and if not found there, in the user environment. Single numeric variables, data frames or matrices are found in the user environment. All entries till the next argument (`data` or `transfo`) are interpreted as explanatory models, and the names of these arguments cannot be abbreviated nor omitted.
`data`	The data frame with the variables used in the formulae in `X`.

`transfo`	Transformation for `Y` (community data) using `decostand`. All alternatives in `decostand` can be used, and those preserving Euclidean metric include `"hellinger"`, `"chi.square"`, `"total"`, `"norm"`.
`scale`	Should the columns of `Y` be standardized to unit variance
`parts`	Number of explanatory tables (circles) displayed.
`labels`	Labels used for displayed fractions. Default is to use the same letters as in the printed output.
`x`	The `varpart` result.
`cutoff`	The values below `cutoff` will not be displayed.
`digits`	The number of significant digits; the number of decimal places is at least one higher.
`...`	Other parameters passed to functions.

Details

The functions partition the variation in `Y` into components accounted for by two to four explanatory tables and their combined effects. If `Y` is a multicolumn data frame or matrix, the partitioning is based on redundancy analysis (RDA, see `rda`), and if `Y` is a single variable, the partitioning is based on linear regression. A simplified, fast version of RDA is used (function `simpleRDA2`). The actual calculations are done in functions `varpart2` to `varpart4`, but these are not intended to be called directly by the user.

The function primarily uses adjusted R squares to assess the partitions explained by the explanatory tables and their combinations, because this is the only unbiased method (Peres-Neto et al., 2006). The raw R squares for basic fractions are also displayed, but these are biased estimates of variation explained by the explanatory table.

The identifiable fractions are designated by lower case alphabets. The meaning of the symbols can be found in the separate document "partionining.pdf" (which can be read using `vegandocs`), or can be displayed graphically using function `showvarparts`.

A fraction is testable if it can be directly expressed as an RDA model. In these cases the printed output also displays the corresponding RDA model using notation where explanatory tables after `|` are conditions (partialled out; see `rda` for details). Although single fractions can be testable, this does not mean that all fractions simultaneously can be tested, since there number of testable fractions is higher than the number of estimated models.

An abridged explanation of the alphabetic symbols for the individual fractions follows, but computational details should be checked in "partitioning.pdf" (readable with `vegandocs`) or in the source code.

With two explanatory tables, the fractions explained uniquely by each of the two tables are `[a]` and `[c]`, and their joint effect is `[b]` following Borcard et al. (1992).

With three explanatory tables, the fractions explained uniquely by each of the three tables are `[a]` to `[c]`, joint fractions between two tables are `[d]` to `[f]`, and the joint fraction between all three tables is `[g]`.

With four explanatory tables, the fractions explained uniquely by each of the four tables are `[a]` to `[d]`, joint fractions between two tables are `[e]` to `[j]`, joint fractions between three variables are `[k]` to `[n]`, and the joint fraction between all four tables is `[o]`.

There is a `plot` function that displays the Venn diagram and labels each intersection (individual fraction) with the adjusted R squared if this is higher than `cutoff`. A helper function `showvarpart` displays the fraction labels.

Value

Function `varpart` returns an object of class `"varpart"` with items `scale` and `transfo` (can be missing) which hold information on standardizations, `tables` which contains names of explanatory tables, and `call` with the function `call`. The function `varpart` calls function `varpart2`, `varpart3` or `varpart4` which return an object of class `"varpart234"` and saves its result in the item `part`. The items in this object are:

`SS.Y`	Sum of squares of matrix `Y`.
`n`	Number of observations (rows).
`nsets`	Number of explanatory tables
`bigwarning`	Warnings on collinearity.
`fract`	Basic fractions from all estimated constrained models.
`indfract`	Invididual fractions or all possible subsections in the Venn diagram (see `showvarparts`).
`contr1`	Fractions that can be found after conditioning on single explanatory table in models with three or four explanatory tables.
`contr2`	Fractions that can be found after conditioning on two explanatory tables in models with four explanatory tables.

Fraction Data Frames

Items `fract`, `indfract`, `contr1` and `contr2` are all data frames with items:

Df	Degrees of freedom of numerator of the F-statistic for the fraction.
R.square	Raw R-squared. This is calculated only for `fract` and this is `NA` in other items.
Adj.R.square	Adjusted R-squared.
Testable	If the fraction can be expressed as a (partial) RDA model, it is directly `Testable`, and this field is `TRUE`. In that case the fraction label also gives the specification of the testable RDA model.

Note

You can use command `vegandocs` to display document "partitioning.pdf" which presents Venn diagrams showing the fraction names in partitioning the variation of Y with respect to 2, 3, and 4 tables of explanatory variables, as well as the equations used in variation partitioning.

The functions frequently give negative estimates of variation. Adjusted R-squares can be negative for any fraction; unadjusted R squares of testable fractions always will be non-negative. Non-testable fractions cannot be found directly, but by subtracting different models, and these subtraction results can be negative. The fractions are orthogonal, or linearly independent, but more complicated or nonlinear dependencies can cause negative non-testable fractions.

The current function will only use RDA in multivariate partitioning. It is much more complicated to estimate the adjusted R-squares for CCA, and unbiased analysis of CCA is not currently implemented.

Author(s)

Pierre Legendre, Departement de Sciences Biologiques, Universite de Montreal, Canada. Adapted to **vegan** by Jari Oksanen.

References

(a) References on variation partioning

Borcard, D., P. Legendre & P. Drapeau. 1992. Partialling out the spatial component of ecological variation. Ecology 73: 1045–1055.

Legendre, P. & L. Legendre. 1998. Numerical ecology, 2nd English edition. Elsevier Science BV, Amsterdam.

(b) Reference on transformations for species data

Legendre, P. and E. D. Gallagher. 2001. Ecologically meaningful transformations for ordination of species data. Oecologia 129: 271–280.

(c) Reference on adjustment of the bimultivariate redundancy statistic

Peres-Neto, P., P. Legendre, S. Dray and D. Borcard. 2006. Variation partioning of species data matrices: estimation and comparison of fractions. Ecology 87: 2614–2625.

See Also

For analysing testable fractions, see `rda` and `anova.cca`. For data transformation, see `decostand`. Function `inertcomp` gives (unadjusted) components of variation for each species or site separately.

Examples

```
data(mite)
data(mite.env)
data(mite.pcnm)

## See detailed documentation:
## Not run:
vegandocs("partition")
## End(Not run)

# Two explanatory matrices -- Hellinger-transform Y
# Formula shortcut "~ ." means: use all variables in 'data'.
mod <- varpart(mite, ~ ., mite.pcnm, data=mite.env, transfo="hel")
mod
showvarparts(2)
plot(mod)
# Alternative way of to conduct this partitioning
# Change the data frame with factors into numeric model matrix
mm <- model.matrix(~ SubsDens + WatrCont + Substrate + Shrub + Topo, mite.env)[,-1]
mod <- varpart(decostand(mite, "hel"), mm, mite.pcnm)
# Test fraction [a] using RDA:
rda.result <- rda(decostand(mite, "hell"), mm, mite.pcnm)
anova(rda.result, step=200, perm.max=200)
```

```
# Three explanatory matrices
mod <- varpart(mite, ~ SubsDens + WatrCont, ~ Substrate + Shrub + Topo,
   mite.pcnm, data=mite.env, transfo="hel")
mod
showvarparts(3)
plot(mod)
# An alternative formulation of the previous model using
# matrices mm1 amd mm2 and Hellinger transformed species data
mm1 <- model.matrix(~ SubsDens + WatrCont, mite.env)[,-1]
mm2 <- model.matrix(~ Substrate + Shrub + Topo, mite.env)[, -1]
mite.hel <- decostand(mite, "hel")
mod <- varpart(mite.hel, mm1, mm2, mite.pcnm)
# Use RDA to test fraction [a]
# Matrix can be an argument in formula
rda.result <- rda(mite.hel ~ mm1 + Condition(mm2) +
   Condition(as.matrix(mite.pcnm)))
anova(rda.result, step=200, perm.max=200)

# Four explanatory tables
mod <- varpart(mite, ~ SubsDens + WatrCont, ~Substrate + Shrub + Topo,
  mite.pcnm[,1:11], mite.pcnm[,12:22], data=mite.env, transfo="hel")
mod
plot(mod)
# Show values for all partitions by putting 'cutoff' low enough:
plot(mod, cutoff = -Inf, cex = 0.7)
```

`vegan-internal` *Internal vegan functions*

Description

Internal vegan functions that are not intended to be called directly, but only within other functions.

Usage

```
ordiGetData(call, env)
ordiParseFormula(formula, data, xlev = NULL, envdepth = 2)
ordiTerminfo(d, data)
ordiArrowMul(x, at = c(0,0), fill = 0.75)
ordiArgAbsorber(..., shrink, origin, scaling, triangular,
                 display, choices, const, FUN)
centroids.cca(x, mf, wt)
permuted.index(n, strata)
pasteCall(call, prefix = "Call:")
```

Details

The description here is only intended for **vegan** developers: these functions are not intended for users, but they only should be used within functions ordiGetData finds the model frame of constraints and conditions in constrained ordination in the defined environment. ordiParseFormula returns a list of three matrices (dependent variables, and model.matrix of constraints and conditions, possibly NULL) needed in constrained ordination. Argument xlev is passed to model.frame and argument envdepth specifies the depth at which the community data (dependent data) is evaluated; default envdepth = 2 evaluates that in the environment of the parent of the calling function, and envdepth = 1 within the calling function (see eval.parent). ordiTermInfo finds the term information for constrained ordination as described in cca.object.

ordiArgAbsorber absorbs arguments of scores function of **vegan** so that these do not cause superfluous warnings in graphical function FUN. If you implement scores functions with new arguments, you should update ordiArgAbsorber.

centroids.cca finds the weighted centroids of variables.

permuted.index creates permuted index of length n possibly stratified within strata. This is the basic **vegan** permutation function that should be replaced with more powerful permuted.index2 in the future releases of **vegan**, and all new functions should use permuted.index2.

pasteCall prints the function call so that it is nicely wrapped in Sweave output.

vegandocs *Display Package Documentation*

Description

Display package documentation using pager or pdfviewer defined in options.

Usage

```
vegandocs(doc = c("NEWS", "ChangeLog", "FAQ-vegan.pdf",
    "intro-vegan.pdf", "diversity-vegan.pdf", "decision-vegan.pdf",
    "partitioning.pdf"))
```

Arguments

doc	The name of the document (partial match, case sensitive).

Note

The function is a kluge, since R does not have this facility (I hope it will come there). Function vignette only works with vignettes.

Author(s)

Jari Oksanen

See Also

vignette.

Examples

```
## Not run:
vegandocs("Change")
## End(Not run)
```

`vegdist` *Dissimilarity Indices for Community Ecologists*

Description

The function computes dissimilarity indices that are useful for or popular with community ecologists. All indices use quantitative data, although they would be named by the corresponding binary index, but you can calculate the binary index using an appropriate argument. If you do not find your favourite index here, you can see if it can be implemented using `designdist`. Gower, Bray–Curtis, Jaccard and Kulczynski indices are good in detecting underlying ecological gradients (Faith et al. 1987). Morisita, Horn–Morisita, Binomial and Chao indices should be able to handle different sample sizes (Wolda 1981, Krebs 1999, Anderson & Millar 2004), and Mountford (1962) and Raup-Crick indices for presence–absence data should be able to handle unknown (and variable) sample sizes.

Usage

```
vegdist(x, method="bray", binary=FALSE, diag=FALSE, upper=FALSE,
        na.rm = FALSE, ...)
```

Arguments

`x`	Community data matrix.
`method`	Dissimilarity index, partial match to `"manhattan"`, `"euclidean"`, `"canberra"`, `"bray"`, `"kulczynski"`, `"jaccard"`, `"gower"`, `"morisita"`, `"horn"`, `"mountford"`, `"raup"` , `"binomial"` or `"chao"`.
`binary`	Perform presence/absence standardization before analysis using `decostand`.
`diag`	Compute diagonals.
`upper`	Return only the upper diagonal.
`na.rm`	Pairwise deletion of missing observations when computing dissimilarities.
`...`	Other parameters. These are ignored, except in `method ="gower"` which accepts `range.global` parameter of `decostand`. .

Details

Jaccard (`"jaccard"`), Mountford (`"mountford"`), Raup–Crick (`"raup"`), Binomial and Chao indices are discussed below. The other indices are defined as:

`euclidean`	$d_{jk} = \sqrt{\sum_i (x_{ij} - x_{ik})^2}$
`manhattan`	$d_{jk} = \sum_i \lvert x_{ij} - x_{ik} \rvert$

`gower`	$d_{jk} = (1/M) \sum_i \frac{\lvert x_{ij}-x_{ik}\rvert}{\max x_i - \min x_i}$ where M is the number of columns (excluding missing values)
`canberra`	$d_{jk} = \frac{1}{NZ} \sum_i \frac{\lvert x_{ij}-x_{ik}\rvert}{x_{ij}+x_{ik}}$ where NZ is the number of non-zero entries.
`bray`	$d_{jk} = \frac{\sum_i \lvert x_{ij}-x_{ik}\rvert}{\sum_i (x_{ij}+x_{ik})}$
`kulczynski`	$d_{jk} = 1 - 0.5(\frac{\sum_i \min(x_{ij},x_{ik})}{\sum_i x_{ij}} + \frac{\sum_i \min(x_{ij},x_{ik})}{\sum_i x_{ik}})$
`morisita`	$d_{jk} = \frac{2\sum_i x_{ij}x_{ik}}{(\lambda_j+\lambda_k)\sum_i x_{ij} \sum_i x_{ik}}$ where $\lambda_j = \frac{\sum_i x_{ij}(x_{ij}-1)}{\sum_i x_{ij} \sum_i (x_{ij}-1)}$
`horn`	Like `morisita`, but $\lambda_j = \sum_i x_{ij}^2 / (\sum_i x_{ij})^2$
`binomial`	$d_{jk} = \sum_i [x_{ij} \log(\frac{x_{ij}}{n_i}) + x_{ik} \log(\frac{x_{ik}}{n_i}) - n_i \log(\frac{1}{2})]/n_i$ where $n_i = x_{ij} + x_{ik}$

Jaccard index is computed as $2B/(1+B)$, where B is Bray–Curtis dissimilarity.

Binomial index is derived from Binomial deviance under null hypothesis that the two compared communities are equal. It should be able to handle variable sample sizes. The index does not have a fixed upper limit, but can vary among sites with no shared species. For further discussion, see Anderson & Millar (2004).

Mountford index is defined as $M = 1/\alpha$ where α is the parameter of Fisher's logseries assuming that the compared communities are samples from the same community (cf. `fisherfit`, `fisher.alpha`). The index M is found as the positive root of equation $\exp(aM) + \exp(bM) = 1 + \exp[(a+b-j)M]$, where j is the number of species occurring in both communities, and a and b are the number of species in each separate community (so the index uses presence–absence information). Mountford index is usually misrepresented in the literature: indeed Mountford (1962) suggested an approximation to be used as starting value in iterations, but the proper index is defined as the root of the equation above. The function `vegdist` solves M with the Newton method. Please note that if either a or b are equal to j, one of the communities could be a subset of other, and the dissimilarity is 0 meaning that non-identical objects may be regarded as similar and the index is non-metric. The Mountford index is in the range $0 \ldots \log(2)$, but the dissimilarities are divided by $\log(2)$ so that the results will be in the conventional range $0 \ldots 1$.

Raup–Crick dissimilarity (`method = "raup"`) is a probabilistic index based on presensec/absence data. It is defined as $1 - prob(j)$, or based on the probability of observing at least j species in shared in compared communities. Legendre & Legendre (1998) suggest using simulations to assess the probability, but the current function uses analytic result from hypergeometric distribution (`phyper`) instead. This probability (and the index) is dependent on the number of species missing in both sites, and adding all-zero species to the data or removing missing species from the data will influence the index. The probability (and the index) may be almost zero or almost one for a wide range of parameter values. The index is nonmetric: two communities with no shared species may have a dissimilarity slightly below one, and two identical communities may have dissimilarity slightly above zero.

Chao index tries to take into account the number of unseen species pairs, similarly as Chao's method in `specpool`. Function `vegdist` implements a Jaccard type index defined as $d_{jk} = U_j U_k/(U_j + U_k - U_j U_k)$, where $U_j = C_j/N_j + (N_k - 1)/N_k \times a_1/(2a_2) \times S_j/N_j$. Here C_j is the total number of individuals in species shared with site k, N is the total number of individuals, a_1 and a_2 are number of species occurring only with one or two individuals in another site, and S_j is the number of individuals in species that occur only with one individual in another site (Chao et al. 2005).

Morisita index can be used with genuine count data (integers) only. Its Horn–Morisita variant is able to handle any abundance data.

Euclidean and Manhattan dissimilarities are not good in gradient separation without proper standardization but are still included for comparison and special needs.

Bray–Curtis and Jaccard indices are rank-order similar, and some other indices become identical or rank-order similar after some standardizations, especially with presence/absence transformation of equalizing site totals with `decostand`. Jaccard index is metric, and probably should be preferred instead of the default Bray-Curtis which is semimetric.

The naming conventions vary. The one adopted here is traditional rather than truthful to priority. The function finds either quantitative or binary variants of the indices under the same name, which correctly may refer only to one of these alternatives For instance, the Bray index is known also as Steinhaus, Czekanowski and Sørensen index. The quantitative version of Jaccard should probably called Ružička index. The abbreviation `"horn"` for the Horn–Morisita index is misleading, since there is a separate Horn index. The abbreviation will be changed if that index is implemented in `vegan`.

Value

Should provide a drop-in replacement for `dist` and return a distance object of the same type.

Note

The function is an alternative to `dist` adding some ecologically meaningful indices. Both methods should produce similar types of objects which can be interchanged in any method accepting either. Manhattan and Euclidean dissimilarities should be identical in both methods. Canberra index is divided by the number of variables in `vegdist`, but not in `dist`. So these differ by a constant multiplier, and the alternative in `vegdist` is in range (0,1). Function `daisy` (package **cluster**) provides alternative implentation of Gower index for mixed data of numeric and class variables (but it works for mixed variables only).

Most dissimilarity indices in `vegdist` are designed for community data, and they will give misleading values if there are negative data entries. The results may also be misleading or `NA` or `NaN` if there are empty sites. In principle, you cannot study species compostion without species and you should remove empty sites from community data.

Author(s)

Jari Oksanen, with contributions from Tyler Smith (Gower index) and Michael Bedward (Raup–Crick index).

References

Anderson, M.J. and Millar, R.B. (2004). Spatial variation and effects of habitat on temperate reef fish assemblages in northeastern New Zealand. *Journal of Experimental Marine Biology and Ecology* 305, 191–221.

Chao, A., Chazdon, R. L., Colwell, R. K. and Shen, T. (2005). A new statistical approach for assessing similarity of species composition with incidence and abundance data. *Ecology Letters* 8, 148–159.

Faith, D. P, Minchin, P. R. and Belbin, L. (1987). Compositional dissimilarity as a robust measure of ecological distance. *Vegetatio* 69, 57–68.

Krebs, C. J. (1999). *Ecological Methodology.* Addison Wesley Longman.

Legendre, P, & Legendre, L. (1998) *Numerical Ecology*. 2nd English Edition. Elsevier.

Mountford, M. D. (1962). An index of similarity and its application to classification problems. In: P.W.Murphy (ed.), *Progress in Soil Zoology*, 43–50. Butterworths.

Wolda, H. (1981). Similarity indices, sample size and diversity. *Oecologia* 50, 296–302.

See Also

Function `designdist` can be used for defining your own dissimilarity index. Alternative dissimilarity functions include `dist` in base R, `daisy` (package **cluster**), and `dsvdis` (package **labdsv**). Function `betadiver` provides indices intended for the analysis of beta diversity.

Examples

```
data(varespec)
vare.dist <- vegdist(varespec)
# Orlóci's Chord distance: range 0 .. sqrt(2)
vare.dist <- vegdist(decostand(varespec, "norm"), "euclidean")
```

`vegemite` *Prints a Compact, Ordered Vegetation Table*

Description

The function prints a compact vegetation table, where species are rows, and each site takes only one column without spaces. The vegetation table can be ordered by explicit indexing, by environmental variables or results from an ordination or cluster analysis.

Usage

```
vegemite(x, use, scale, sp.ind, site.ind, zero=".", select ,...)
coverscale(x, scale=c("Braun.Blanquet", "Domin", "Hult", "Hill", "fix","log"),
              maxabund)
```

Arguments

`x`	Vegetation data.
`use`	Either a vector, or an object from `cca`, `decorana` *etc.* or `hclust` or a `dendrogram` for ordering sites and species.
`sp.ind`	Species indices.

`site.ind`	Site indices.
`zero`	Character used for zeros.
`select`	Select a subset of sites. This can be a logical vector (`TRUE` for selected sites), or a vector of indices of selected sites. The order of indices does not influence results, but you must specify `use` or `site.ind` to reorder sites.
`scale`	Cover scale used (can be abbreviated).
`maxabund`	Maximum abundance used with `scale = "log"`. Data maximum in the `select`ed subset will be usd if this is missing.
`...`	Arguments passed to `coverscale` (i.e., `maxabund`).

Details

The function prints a traditional vegetation table. Unlike in ordinary data matrices, species are used as rows and sites as columns. The table is printed in compact form: only one character can be used for abundance, and there are no spaces between columns. Species with no occurrences are dropped from the table.

The parameter `use` can be a vector or an object from `hclust`, a `dendrogram` or any ordination result recognized by `scores` (all ordination methods in **vegan** and some of those not in **vegan**). If `use` is a vector, it is used for ordering sites. If `use` is an object from ordination, both sites and species are arranged by the first axis. When `use` is an object from `hclust` or a `dendrogram`, the sites are ordered similarly as in the cluster dendrogram. If ordination methods provide species scores, these are used for ordering species. In all cases where species scores are missing, species are ordered by their weighted averages (`wascores`) on site scores. There is no natural, unique ordering in hierarchic clustering, but in some cases species are still nicely ordered (please note that you can `reorder.dendrogram` to have such a natural order). Alternatively, species and sites can be ordered explicitly giving their indices or names in parameters `sp.ind` and `site.ind`. If these are given, they take precedence over `use`. A subset of sites can be displayed using argument `select`, but this cannot be used to order sites, but you still must give `use` or `site.ind`.

If `scale` is given, `vegemite` calls `coverscale` to transform percent cover scale or some other scales into traditional class scales used in vegetation science (`coverscale` can be called directly, too). Braun-Blanquet and Domin scales are actually not strict cover scales, and the limits used for codes `r` and `+` are arbitrary. Scale `Hill` may be inappropriately named, since Mark O. Hill probably never intended this as a cover scale. However, it is used as default 'cut levels' in his `TWINSPAN`, and surprisingly many users stick to this default, and this is a *de facto* standard in publications. All traditional scales assume that values are cover percentages with maximum 100. However, non-traditional alternative `log` can be used with any scale range. Its class limits are integer powers of 1/2 of the maximum (argument `maxabund`), with `+` used for non-zero entries less than 1/512 of the maximum (`log` stands alternatively for logarithmic or logical). Scale `fix` is intended for 'fixing' 10-point scales: it truncates scale values to integers, and replaces 10 with `X` and positive values below 1 with `+`.

Value

The function is used mainly to print a table, but it returns (invisibly) a list with items:

`species`	Ordered species indices
`sites`	Ordered site indices

These items can be used as arguments `sp.ind` and `site.ind` to reproduce the table. In addition to the proper table, the function prints the numbers of species and sites and the name of the used cover scale at the end.

Note

This function was called `vegetab` in older versions of `vegan`. The new name was chosen because the output is so compact (and to avoid confusion with the `vegtab` function in the **labdsv** package).

Author(s)

Jari Oksanen

References

The cover scales are presented in many textbooks of vegetation science; I used:

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

See Also

`cut` and `approx` for making your own 'cover scales', `wascores` for weighted averages.

Examples

```
data(varespec)
## Print only more common species
freq <- apply(varespec > 0, 2, sum)
vegemite(varespec, scale="Hult", sp.ind = freq > 10)
## Order by correspondence analysis, use Hill scaling and layout:
dca <- decorana(varespec)
vegemite(varespec, dca, "Hill", zero="-")
## Show one class from cluster analysis, but retain the ordering above
clus <- hclust(vegdist(varespec))
cl <- cutree(clus, 3)
sel <- vegemite(varespec, use=dca, select = cl == 3, scale="Br")
# Re-create previous
vegemite(varespec, sp=sel$sp, site=sel$site, scale="Hult")
```

`wascores` *Weighted Averages Scores for Species*

Description

Computes Weighted Averages scores of species for ordination configuration or for environmental variables.

Usage

```
wascores(x, w, expand=FALSE)
eigengrad(x, w)
```

Arguments

`x`	Environmental variables or ordination scores.
`w`	Weights: species abundances.
`expand`	Expand weighted averages so that they have the same weighted variance as the corresponding environmental variables.

Details

Function `wascores` computes weighted averages. Weighted averages 'shrink': they cannot be more extreme than values used for calculating the averages. With `expand = TRUE`, the function 'dehsrinks' the weighted averages by making their biased weighted variance equal to the biased weighted variance of the corresponding environmental variable. Function `eigengrad` returns the inverses of squared expansion factors or the attribute `shrinkage` of the `wascores` result for each environmental gradient. This is equal to the constrained eigenvalue of `cca` when only this one gradient was used as a constraint, and describes the strength of the gradient.

Value

Function `wascores` returns a matrix where species define rows and ordination axes or environmental variables define columns. If `expand = TRUE`, attribute `shrinkage` has the inverses of squared expansion factors or `cca` eigenvalues for the variable. Function `eigengrad` returns only the `shrinkage` attribute.

Author(s)

Jari Oksanen

See Also

`isoMDS`, `cca`.

Examples

```
data(varespec)
data(varechem)
library(MASS)  ## isoMDS
vare.dist <- vegdist(wisconsin(varespec))
vare.mds <- isoMDS(vare.dist)
vare.points <- postMDS(vare.mds$points, vare.dist)
vare.wa <- wascores(vare.points, varespec)
plot(scores(vare.points), pch="+", asp=1)
text(vare.wa, rownames(vare.wa), cex=0.8, col="blue")
## Omit rare species (frequency <= 4)
freq <- apply(varespec>0, 2, sum)
plot(scores(vare.points), pch="+", asp=1)
text(vare.wa[freq > 4,], rownames(vare.wa)[freq > 4],cex=0.8,col="blue")
## Works for environmental variables, too.
wascores(varechem, varespec)
## And the strengths of these variables are:
eigengrad(varechem, varespec)
```

`wcmdscale` *Weighted Classical (Metric) Multidimensional Scaling*

Description

Weighted classical multidimensional scaling, also known as weighted *principal coordinates analysis*.

Usage

```
wcmdscale(d, k, eig = FALSE, add = FALSE, x.ret = FALSE, w)
```

Arguments

`d`	a distance structure such as that returned by `dist` or a full symmetric matrix containing the dissimilarities.
`k`	the dimension of the space which the data are to be represented in; must be in $\{1, 2, \ldots, n-1\}$. If missing, all dimensions with above zero eigenvalue.
`eig`	indicates whether eigenvalues should be returned.
`add`	logical indicating if an additive constant $c*$ should be computed, and added to the non-diagonal dissimilarities such that all $n-1$ eigenvalues are non-negative. **Not implemented**.
`x.ret`	indicates whether the doubly centred symmetric distance matrix should be returned.
`w`	Weights of points.

Details

Function `wcmdscale` is based on function `cmdscale` (package **stats** of base R), but it uses point weights. Points with high weights will have a stronger influence on the result than those with low weights. Setting equal weights `w = 1` will give ordinary multidimensional scaling.

Value

If `eig = FALSE` and `x.ret = FALSE` (default), a matrix with `k` columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

Otherwise, an object of class `wcmdscale` list containing the following components.

`points`	a matrix with `k` columns whose rows give the coordinates of the points chosen to represent the dissimilarities.
`eig`	the $n-1$ eigenvalues computed during the scaling process if `eig` is true.
`x`	the doubly centred and weighted distance matrix if `x.ret` is true.
`weights`	Weights.

References

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

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979). Chapter 14 of *Multivariate Analysis*, London: Academic Press.

See Also

`cmdscale`. Also `isoMDS` and `sammon` in package **MASS**.

Examples

```
## Correspondence analysis as a weighted principal coordinates
## analysis of Euclidean distances of Chi-square transformed data
data(dune)
rs <- rowSums(dune)/sum(dune)
d <- dist(decostand(dune, "chi"))
ord <- wcmdscale(d, w = rs, eig = TRUE)
## Ordinary CA
ca <- cca(dune)
## Eigevalues are numerically similar
ca$CA$eig - ord$eig
## Configurations are similar when site scores are scaled by
## eigenvalues in CA
procrustes(ord, ca, choices=1:19, scaling = 1)
plot(procrustes(ord, ca, choices=1:2, scaling=1))
```

Index