The vegan Package

September 26, 2005

Title Community Ecology Package

Version 1.6-10

Date September 26, 2005

Author Jari Oksanen, Roeland Kindt, Bob O'Hara

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

Description Ordination methods and other useful functions for community and vegetation ecologists.

License GPL2

URL http://cc.oulu.fi/~jarioksa/

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.

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.

Examples

```
data(BCI)
```

`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.

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

`dist` and `vegdist` for obtaining dissimilarities, and `rank` for ranking real values. For comparing dissimilarities against continuous variables, see `mantel`.

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=10000, ...)
permutest.cca(x, permutations=100, model=c("direct", "reduced","full"), 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.
`...`	Parameters to permutest.cca.
`permutations`	Number of permutations for assessing significance of constraints.
`model`	Permutation model (partial match).
`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.

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"`, 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. 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.

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.

Author(s)

Jari Oksanen

References

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

See Also

`cca`, `rda`, `capscale`.

Examples

```
data(varespec)
data(varechem)
vare.cca <- cca(varespec ~ Al + P + K, varechem)
anova(vare.cca)
permutest.cca(vare.cca)
## Test for adding variable N to the previous model:
anova(cca(varespec ~ N + Condition(Al + P + K), varechem), step=40)
```

---

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), ...)
## 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.test`.
`index`	The dissimilarity index used for community data in `vegdist`.
`upto`	Maximum number of parameters in studied subsets.
`formula, data`	Model `formula` and data.
`...`	Other parameters passed to function.

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 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.

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.

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)
```

---

capscale *[Partial] Constrained Analysis of Principal Coordinates*

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 may be useful with other dissimilarity measures, since Euclidean distances inherent in rda are generally poor with community data

Usage

```
capscale(formula, data, distance = "euclidean", comm = NULL, add =
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. RHS defines the constraints. The constraints can be continuous or factors, they can be transformed within the formula, and they can have interactions as in typical formula. The RHS can have a special term Condition that defines variables "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 underlying Principal Co-ordinates Analysis (see cmdscale for details).
...	Other parameters passed to rda.

Details

The 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) analyses 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:

1. Anderson & Willis used orthonormal solution of `cmdscale`, whereas `capscale` uses axes weighted by corresponding eigenvalues, so that the ordination distances are 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.

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`, `summary.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 or equal to 4 (allowing for `stepacross`), this division is undone in `capscale` and original dissimilarities are used. The inertia is called as `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.

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)
```

---

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, ...)
## S3 method for class 'cca':
summary(object, scaling=2, axes=6, digits, ...)
```

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.
`object`	A `cca` result object.
`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. Unscaled raw scores stored in the result can be accessed with `scaling = 0`.
`axes`	Number of axes in summaries.
`digits`	Number of digits in output.
`scale`	Scale species to unit variance (like correlations do).
`...`	Other parameters for `print` or `plot` 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 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 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 matrix interface, the community data matrix `X` must be given, but any other data matrix can 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 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 single variable constraints and conditions.

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 a common curve artefact 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 allow curve artefact (and are difficult to interpret), and 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.

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. The `summary` method lists all species and site scores, and results may be very long. 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

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 desribed 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` function with its 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`. Functions `CAIV` (library `CoCoAn`) and `cca` (library `ade4`) provide alternative implementations of CCA (these are internally quite different). Function `capscale` is a non-Euclidean generalization of `rda`.

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)
```

`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` 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 item `QR`.

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 component.

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 `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.

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"), ...)
downweight(veg, fraction = 5)
## S3 method for class 'decorana':
scores(x, display=c("sites","species"), choices =1:4, origin=TRUE, ...)
```

Arguments

`veg`	Community data matrix.
`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.
`fraction`	Abundance fraction where downweighting begins.
`...`	Other parameters for `plot` function.

Details

In late 1970s, correspondence analysis became the method of choice for ordination in vegetation science, since it seemed to be able to cope with non-linear species responses better than principal components analysis. However, even correspondence analysis produced 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 the successive axes are made non-correlated, but detrending should remove all systematic dependence between axes. Detrending is made 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 works actually by twisting the ordination space, so that the results look non-curved in two-dimensional projections ('lolly paper effect'). As a result, the points have usually an easily recognized triangle or diamond shaped pattern, obviously as a detrending artefact. 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.

Function `summary` prints the ordination scores, possible prior weights used in downweighting, and the marginal totals after applying these weights. Function `plot` 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 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 always 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

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

Note

Function `decorana` uses the central numerical engine of the original Fortran code (which is in 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 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`, `CAIV` and `ca` as well.

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

Author(s)

Mark O. Hill wrote the original Fortran code, 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. 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 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` *Standardizaton 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 matrix.
`method`	Standardization method.
`MARGIN`	Margin, if default is not acceptable.
`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 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 ... 1 (default `MARGIN = 2`). If all values are constant, they will be transformed to 0.
- `standardize`: scale into zero mean and unit variance (default `MARGIN = 2`).
- `pa`: scale into 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 Euclidean distance, the matrix 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 non-sense 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.

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: Similar but not identical to Correspondence Analysis.
sptrans <- decostand(varespec, "chi.square")
plot(procrustes(rda(sptrans), cca(varespec)))
```

`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 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 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 variable with highest eigenvalues is not necessarily the most significant one in permutation tests in `anova.cca`). If you also have multiclass factors, the value of `k` will have a capricious effect in 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`.

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 for the added variable
anova(cca(dune ~ Moisture, dune.env))
# ANOVA for the next candidate variable that was not added
anova(cca(dune ~ Condition(Moisture) + Management, dune.env), perm.max=1000)
```

`distconnected` *Connectedness and Minimum Spanning Tree for 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. Function `spantree` finds a minimum spanning tree connecting all points, but disregarding dissimilarities that are at or above the threshold or `NA`.

Usage

```
distconnected(dis, toolong = 1, trace = TRUE)
no.shared(x)
spantree(dis, toolong = 0)
```

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.
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 NAs. Function distconnected will find such subsets in dissimilarity matrices. The function will return a grouping vector that can be used for subsetting 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 NAs.

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 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 NAs 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. The results of spantree can be overlaid onto an ordination diagram using function ordispantree.

Function distconnected uses depth-first search (Sedgewick 1990). Function spantree uses Prim's method implemented as priority-first search for dense graphs (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. Function spantree returns 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 ordispantree.

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`, `ordispantree` for displaying results of `spantree`, and `hclust` or `agnes` for single linkage clustering.

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)
ord <- cmdscale(dis) ## metric MDS
gr <- distconnected(dis, toolong=0.4)
tr <- spantree(dis, toolong=0.4)
ordiplot(ord, type="n")
ordispantree(ord, tr, col="red", lwd=2)
points(ord, cex=1.3, pch=21, col=1, bg = gr)
# 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, Rényi, Hill 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)
renyi(x, scales=c(0,0.25,0.5,1,2,4,8,16,32,64,Inf), hill = FALSE)
fisher.alpha(x, MARGIN = 1, se = FALSE, ...)
specnumber(x, MARGIN = 1)
```

Arguments

`x`	Community data matrix.
`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.
`se`	Estimate standard errors.
`scales`	Scales of Rényi diversity.
`hill`	Calculate Hill numbers.
`...`	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 $1/D$.

Shannon and Simpson indices are both 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).

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). 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).

Function `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 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 more grandiose stories 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, 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

Vector of diversity indices or rarefied species richness values. With option `se = TRUE`, function `rarefy` returns a 2-row matrix with rarefied richness (`S`) and its standard error (`se`). Function `renyi` returns a data frame of selected indices. 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, Roeland Kindt ⟨r.kindt@cgiar.org⟩ (`renyi`) 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.

Hill, M.O. (1973). Diversity and evenness: a unifying notation and its consequences. *Ecology* 54, 427–473.

Hurlbert, S.H. (1971). The nonconcept of species diversity: a critique and alternative parameters. *Ecological Monographs* 54, 187–211.

Tóthmérész, B. (1995). Comparison of different methods for diversity ordering. *Journal of Vegetation Science* 6, 283–290.

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

A1 a numeric vector of thickness of A1 horizon.

Moisture an ordered factor with levels

Moisture `1` < `2` < `4` < `5`

Management a factor with levels

Management `BF`: Biological Farming

Management `HF`: Hobby Farming

Management `NM`: Nature Conservation Management

Management `SF`: Standard Farming

Use an ordered factor of landuse with levels

Use `Hayfield` < `Haypastu` < `Pasture`

Manure an ordered factor with levels

Manure `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*. Pudog, Wageningen.

Examples

```
data(dune)
```

---

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

Description

The function fits environmental vectors or factors onto an ordination. The projection of points onto vectors have maximum correlations 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. This means that sites with lower totals will have lower weights. 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 error, 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` or `rad.veil`.

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.

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.

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)
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.
`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, or 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 site 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 `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))
```

`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 ststistics on the goodness of fit. The correlation based on stress S is 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))
```

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

Description

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

Usage

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

Arguments

`x`	Numeric vector to be plotted.
`cex`	Size of the labels.
`label`	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 draws a plot and returns nothing useful.

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"), label="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 variables had to be stored into 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, and `anosim` 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, expand = TRUE, trace = 1, plot = FALSE,
        previous.best, ...)
## 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, ...)
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.
`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.
`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.
`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 $0 \ldots 1$, 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.

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.

Note

Function `metaMDS` is a simple wrapper for `isoMDS` and some support functions. You can also call parts of the function 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")
```

`ordihull` *Add Graphical Items to Ordination Diagrams*

Description

Functions to add convex hulls, arrows, line segments, regular grids of points, 'spider' graphs, ellipses, cluster dendrogram or spanning trees 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, ...)
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), ...)
ordispantree(ord, tree, display = "sites", ...)
```

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 `polygon`s 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.
`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.
`tree`	Structure defining a spanning tree. This can be a result of `spantree` or a vector giving the child node of each parent omitting the first point. Values `NA` means that there is no link from the corresponding parent.
`...`	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.

Function `ordispantree` overlays a (minimum) spanning tree onto ordination. It needs a result from `spantree` or a vector listing children of each parent, starting from second (i.e., omitting the first: the number of links is one less number of points). Missing links are denoted as `NA`. For an example, see `spantree`.

Note

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

Author(s)

Jari Oksanen

See Also

The function 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)
plot(mod, type="n")
ordihull(mod, Moisture)
ordispider(mod, col="red")
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", level=0.95, lwd=2, col="blue")
## End(Not run)
```

`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 ordinatin (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)
```

`ordisurf` *Smooths Variables and Plots Contours 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), ...)
```

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.
`...`	Other graphical parameters.

Details

Function `ordisurf` fits a smooth surface using thinplate spline fitting in `gam`, and interpolates the fitted values into a regular grid using `interp`. Finally, it plots the results either over an existing ordination diagram or draws a new plot with sample plots and fitted contours. 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. This means that sites with lower totals will have lower weights. 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 function requires libraries `mgcv` (`gam`) and `akima` (`interp`). In fact, it is a very primitive wrapper for these.

The default is to use thinplate splines. These make sense in ordination as they have equal smoothing in all directions and are rotation invariant. However, they seem to fail badly in some case, and then separate spline smoothing may be used.

Author(s)

Dave Roberts and Jari Oksanen

See Also

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

Examples

```
## The examples are not run by `example(ordisurf)' because they need
## libraries `mgcv' and `akima' which may not exist in every system.
## Not run:
data(varespec)
data(varechem)
library(MASS)
vare.dist <- vegdist(varespec)
vare.mds <- isoMDS(vare.dist)
attach(varespec)
attach(varechem)
ordisurf(vare.mds, Baresoil, xlab="Dim1", ylab="Dim2")
## Total cover of reindeer lichens
ordisurf(vare.mds, Cla.ste+Cla.arb+Cla.ran, xlab="Dim1", ylab="Dim2")
## 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, tcex = 0.7,
    pcex, tcol = 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.
`tcex, pcex`	Text and point sizes, see `plot.default`..
`tcol, 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 `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`).

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")
```

`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, ...)
## 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, ...)
```

Arguments

`x`	A `cca` result object.
`choices`	Axes shown.
`display`	Scores shown. These must some of the alternatives `sp` for species scores, `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.
`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.
`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.
`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.
`...`	Other parameters for plotting 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 thus 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 with `points.cca` and `text.cca` the user must give the expansion factor in `mul.arrow`. The behaviour is still more peculiar with `display="cn"` which requests centroids of levels of `factor` variables (these are available only if there were factors and a formula interface was used in `cca` or `rda`). With this option, biplot arrows are plotted in addition to centroids in cases which do not have a centroid: Continuous variables are presented with arrows and ordered factors with 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`.

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

Option `display="cn"` (centroids and biplot arrows) may become the default instead of the current `display="bp"` in the future version.

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)
```

`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"), ...)
## 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: `"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. Function `residuals` gives the approximation of the original data from the unconstrained ordination. The `fitted.cca` and `residuals.cca` function both have 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 sccores to existing ordinations. The function returns (weighted) orthornormal 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.

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.

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

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*

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, ...)
## 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`.
`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)
```

`radfit` *Rank – Abundance or Dominance / Diversity Models*

Description

Functions construct rank – abundance or dominance / diversity or Whittaker plots and fit preemption, log-Normal, veiled 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.preempt(x, family = poisson, ...)
rad.lognormal(x, family = poisson, ...)
rad.veil(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 in the community. These plots are supposed to be effective in analysing types of abundance distributions in communities. These functions fit some of the most popular models 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.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. In addition there is a fixed scaling parameter J which is the total abundance. 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.veil` is similar, but it assumes that only a proportion `veil` of most common species were observed in the community, the rest being too rare or scanty

to occur in a sample plot of this size (but would occur in a larger plot). Function `rad.zipf` fits the Zipf model $a_r = Jp_1r^\gamma$ 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 great histories 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. Veiled log-Normal and Zipf – Mandelbrot add one nonlinear parameter, and these two models are 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. 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. 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, `rad.veil` often tends to `veil = 0` meaning that none of the community is present, and the function prints an error message `Error: NA/NaN/Inf in foreign function call (arg 1)`. The error is caught and `NA` are 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.

Author(s)

Jari Oksanen

References

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.veil(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`.

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)
```

`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, display=c("sites", "species"), choices, ...)
```

Arguments

`x`	An ordination result.
`display`	Partial match to access scores for `sites` or `species`.
`choices`	Ordination axes. If missing, returns all axes.
`...`	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))
```

---

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, ...)
## 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, ...)
## 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".
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.
...	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 of Kindt (2003), and its standard deviation. 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 average number of sites corresponding to a certain number of 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.

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.*

See Also

`rarefy`. 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	$S_P = S_0 + \frac{a_1^2}{(a_2+1)} + \frac{a_1 a_2}{2(a_2+1)^2}$
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 and ACE. The Chao estimator is identical to the one above, except that a_i refers to number of species with abundance i instead of incidence. The ACE is defined as:

$$\begin{array}{ll} \text{ACE} & S_P = S_{abund} + \frac{S_{rare}}{C_{ace}} + \frac{a_1}{C_{ace}} \gamma^2_{ace} \\ \text{where} & C_{ace} = 1 - \frac{a_1}{N_{rare}} \\ & \gamma^2_{ace} = \max \left[\frac{S_{rare} \sum_{i=1}^{10} i(i-1) a_i}{C_{ace} N_{rare} (N_{rare}-1)} - 1, 0 \right] \end{array}$$

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`.

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 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 in several cases they are 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 NAs 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.

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)
```

---

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)
```

---

vegan-internal *Internal vegan functions*

Description

Internal vegan functions.

Usage

```
ordiParseFormula(formula, data, xlev = NULL)
ordiTerminfo(d, data)
centroids.cca(x, mf, wt)
permuted.index(n, strata)
spider.cca(x, ...)
```

Details

These are not to be called by the user. Function `spider.cca` was replaced with `ordispider` and will be removed in the future.

`vegdist` *Dissimilarity Indices for Community Ecologists*

Description

The function computes dissimilarity indices that are useful for or popular with community ecologists. Gower, Bray–Curtis, Jaccard and Kulczynski indices are good in detecting underlying ecological gradients (Faith et al. 1987). Morisita, Horn–Morisita and Binomial 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"` or `"binomial"`.
`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"`) and Binomial 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`	$d_{jk} = \sum_i \frac{\lvert x_{ij} - x_{ik} \rvert}{\max x_i - \min x_i}$
`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`	$d_{jk} = 1 - 0.5\left(\frac{\sum_i \min(x_{ij}, x_{ik})}{\sum_i x_{ij}} + \frac{\sum_i \min(x_{ij}, x_{ik})}{\sum_i x_{ik}}\right)$
`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 (1978) 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.

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 quantitive version of Jaccard should probably called Ruzicka index (but spelled with characters that cannot be shown here). 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.

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

`decostand`, `dist`, `rankindex`, `isoMDS`, `stepacross`, `daisy`, `dsvdis`.

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=".")
coverscale(x, scale=c("Braun.Blanquet", "Domin", "Hult", "Hill", "fix", "log"))
```

Arguments

`x`	Vegetation data.
`use`	Either a vector or an object from `cca`, `decorana` *etc.* or `hclust` for ordering sites and species.
`sp.ind`	Species indices.
`site.ind`	Site indices.
`zero`	Character used for zeros.
`scale`	Cover scale used (can be abbreviated).

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.

The parameter `use` can be a vector or an object from `hclust`, a `dendrogram` or any ordination result recognized by `scores`. 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`, 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. 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`.

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 so 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 observed maximum in the data, with `+` used for non-zero entries less than 1/512 of data 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:

`spec`	Ordered species indices.
`sites`	Ordered site indices.

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="-")
```

---

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)
```

Index