Package: vegan (via r-universe)

October 21, 2026

Title Community Ecology Package

Version 2.8-0

Depends permute (>= 0.9-0), R (>= 4.1.0)

Suggests knitr, markdown, tinytest

Imports MASS, cluster, lattice, mgcv, parallel

VignetteBuilder utils, knitr

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

License GPL-2

BugReports https://github.com/vegandevs/vegan/issues

URL https://vegandevs.github.io/vegan/, https://github.com/vegandevs/vegan

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

Date/Publication 2026-10-21 08:24:04 UTC

RemoteUrl https://github.com/vegandevs/vegan

RemoteRef HEAD

RemoteSha c9939a9ac2c52933108bef98920f0400a1e59ad5

Contents

vegan-package 4
add1.cca 5
adipart 7
adonis 11
anosim 14
anova.cca 16
avgdist 19
BCI 21
beals 23
betadisper 25

betadiver . . . 31
bgdispersal . . . 33
bioenv . . . 35
biplot.rda . . . 38
cascadeKM . . . 40
cca . . . 43
cca.object . . . 47
CCorA . . . 51
clamtest . . . 54
commsim . . . 56
contribdiv . . . 62
dbrda . . . 64
decorana . . . 68
decostand . . . 72
designdist . . . 75
deviance.cca . . . 79
dispindmorisita . . . 81
dispweight . . . 82
distconnected . . . 84
diversity . . . 86
dune . . . 88
dune.taxon . . . 89
eigenvals . . . 90
envfit . . . 92
eventstar . . . 96
fisherfit . . . 99
goodness.cca . . . 102
goodness.metaMDS . . . 104
indpower . . . 106
influence.cca . . . 107
isomap . . . 110
kendall.global . . . 113
linestack . . . 115
make.cepnames . . . 117
mantel . . . 118
mantel.correlog . . . 121
MDSaddpoints . . . 124
MDSrotate . . . 125
metaMDS . . . 127
mite . . . 133
monoMDS . . . 135
MOStest . . . 138
mrpp . . . 141
mso . . . 145
multipart . . . 148
nestedtemp . . . 150
nobs.cca . . . 154
nullmodel . . . 154

oecosimu . . . 158
optspace . . . 163
ordiarrows . . . 164
ordiArrowTextXY . . . 166
ordihull . . . 168
ordilabel . . . 172
ordiplot . . . 173
ordipointlabel . . . 176
ordistep . . . 178
ordisurf . . . 181
orditorp . . . 186
pcnm . . . 188
permat . . . 190
permustats . . . 194
permutations . . . 197
permutest.betadisper . . . 200
plot.cca . . . 202
prc . . . 207
predict.cca . . . 209
procrustes . . . 213
pyrifos . . . 217
radfit . . . 218
rankindex . . . 221
rarefy . . . 223
raupcrick . . . 225
read.cep . . . 227
renyi . . . 229
reorder.hclust . . . 231
RsquareAdj . . . 233
scores . . . 234
screeplot.cca . . . 235
simper . . . 238
simulate.rda . . . 240
sipoo . . . 242
spantree . . . 243
specaccum . . . 245
specpool . . . 250
sppscores . . . 253
SSarrhenius . . . 255
stepacross . . . 257
stressplot.wcmdscale . . . 259
taxondive . . . 261
tolerance . . . 263
treedive . . . 265
tsallis . . . 267
varespec . . . 269
varpart . . . 270
vegan-deprecated . . . 275

vegdist 276
vegemite 282
wascores 285
wcmdscale 288

Index **291**

`vegan-package` *Community Ecology Package: Ordination, Diversity and Dissimilarities*

Description

The **vegan** package provides tools for descriptive community ecology. It has most basic functions of diversity analysis, community ordination and dissimilarity analysis. Most of its multivariate tools can be used for other data types as well.

Details

The functions in the **vegan** package contain tools for diversity analysis, ordination methods and tools for the analysis of dissimilarities. Together with the **labdsv** package, the **vegan** package provides most standard tools of descriptive community analysis. Package **ade4** provides an alternative comprehensive package, and several other packages complement **vegan** and provide tools for deeper analysis in specific fields. Package **BiodiversityR** provides a GUI for a large subset of **vegan** functionality.

The **vegan** package is developed at GitHub (`https://github.com/vegandevs/vegan/`). GitHub provides up-to-date information and forums for bug reports.

Most important changes in **vegan** documents can be read with `news(package="vegan")` and vignettes can be browsed with `browseVignettes("vegan")`. The vignettes include a **vegan** FAQ, discussion on design decisions, short introduction to ordination and discussion on diversity methods.

To see the preferable citation of the package, type `citation("vegan")`.

Author(s)

The **vegan** development team is Jari Oksanen, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, Peter R. Minchin, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, Helene Wagner. Many other people have contributed to individual functions: see credits in function help pages.

Examples

```
### Example 1: Unconstrained ordination
## NMDS
data(varespec, varechem)
ord <- metaMDS(varespec)
plot(ord, optimize = TRUE, type = "t")
## Fit environmental variables
```

```
ef <- envfit(ord, varechem)
ef
plot(ef, p.max = 0.05)
### Example 2: Constrained ordination (RDA)
## The example uses formula interface to define the model
data(dune, dune.env)
## No constraints: PCA
mod0 <- rda(dune ~ 1, dune.env)
mod0
plot(mod0, spe.par = list(arrows = TRUE))
## All environmental variables: Full model
mod1 <- rda(dune ~ ., dune.env)
mod1
plot(mod1)
## Automatic selection of variables by permutation P-values
mod <- ordistep(mod0, scope=formula(mod1))
mod
plot(mod, spe.par = list(optimize = TRUE))
## Permutation test for all variables
anova(mod)
## Permutation test of "type III" effects, or significance when a term
## is added to the model after all other terms
anova(mod, by = "margin")
## Plot only sample plots, use different symbols and draw SD ellipses
## for Management classes
plot(mod, display = "sites", type = "n")
with(dune.env, points(mod, disp = "si", pch = as.numeric(Management)))
with(dune.env, legend("topleft", levels(Management), pch = 1:4,
  title = "Management"))
with(dune.env, ordiellipse(mod, Management, label = TRUE))
## add fitted surface and direction arrow of diversity to the model
ordisurf(mod, diversity(dune), add = TRUE)
plot(envfit(mod ~ diversity(dune), permutations=0))
### Example 3: analysis of dissimilarites a.k.a. non-parametric
### permutational anova
adonis2(dune ~ ., dune.env, by = "margin")
adonis2(dune ~ Management + Moisture, dune.env, by = "term")
```

---

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

Description

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

Usage

```
## S3 method for class 'cca'
add1(object, scope, test = c("none", "permutation", "F"),
    permutations = how(nperm=199), ...)
```

```
## S3 method for class 'cca'
drop1(object, scope, test = c("none", "permutation", "F"),
    permutations = how(nperm=199), ...)
```

Arguments

`object`	A constrained ordination object from `cca`, `rda`, `dbrda` or `capscale`.
`scope`	A formula giving the terms to be considered for adding or dropping; see `add1` for details.
`test`	Should a `"permutation"` or parametric `"F"` test be added using `anova.cca`. `test = "F"` can be only used with `rda` models.
`permutations`	a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
`...`	Other arguments passed to `add1.default`, `drop1.default`, and `anova.cca`.

Details

With argument `test = "none"` the functions will only call `add1.default` or `drop1.default`. With other choice the functions will add test results from `anova.cca`, either a permutation test or a parameteric F-test (only with `rda` models). Function `drop1.cca` will call `anova.cca` with argument `by = "margin"`. Function `add1.cca` will implement a test for single term additions that is not directly available in `anova.cca`.

Functions are used implicitly in `step`, `ordiR2step` and `ordistep`. The `deviance.cca` and `deviance.rda` used in `step` have no firm basis, and setting argument `test = "permutation"` may help in getting useful insight into validity of model building. Function `ordistep` calls alternately `drop1.cca` and `add1.cca` with argument `test = "permutation"` and selects variables by their permutation P-values. Meticulous use of `add1.cca` and `drop1.cca` will allow more judicious model building.

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

Value

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

Author(s)

Jari Oksanen

See Also

`add1`, `drop1` and `anova.cca` for basic methods. Functions `deviance.cca` and `extractAIC.cca` are used to produce the other arguments than test results in the output. `add1` and `drop1` are the basic functions in automatic model building with `step`, `ordistep` and `ordiR2step`.

Examples

```
data(dune, dune.env)

## Automatic model building based on AIC but with permutation tests
step(cca(dune ~  1, dune.env), formula(cca(dune ~ ., dune.env)),
  test="permutation")
## vegan ordistep does the same, but based on P-values
ordistep(cca(dune ~  1, dune.env), cca(dune ~ ., dune.env))

## Manual model building using parametric F test
## define the maximal model for scope
mbig <- rda(dune ~  ., dune.env)
## define an empty model to start with
m0 <- rda(dune ~ 1, dune.env)
## manual selection and updating
add1(m0, scope=formula(mbig), test="F")
m0 <- update(m0, . ~ . + Management)
add1(m0, scope=formula(mbig), test="F")
m0 <- update(m0, . ~ . + Moisture)
## included variables still significant?
drop1(m0, test="F")
add1(m0, scope=formula(mbig), test="F")
```

---

adipart *Additive Diversity Partitioning and Hierarchical Null Model Testing*

Description

In additive diversity partitioning, mean values of alpha diversity at lower levels of a sampling hierarchy are compared to the total diversity in the entire data set (gamma diversity). In hierarchical null model testing, a statistic returned by a function is evaluated according to a nested hierarchical sampling design (hiersimu).

Usage

```
adipart(...)
## Default S3 method:
adipart(y, x, index, weights=c("unif", "prop"),
    relative = FALSE, nsimul=99, method = "r2dtable", ...)
## S3 method for class 'formula'
adipart(formula, data, index=c("richness", "shannon", "simpson"),
    weights=c("unif", "prop"), relative = FALSE, nsimul=99,
    method = "r2dtable", ...)

hiersimu(...)
## Default S3 method:
hiersimu(y, x, FUN, location = c("mean", "median"),
    relative = FALSE, drop.highest = FALSE, nsimul=99,
```

```
    method = "r2dtable", ...)
## S3 method for class 'formula'
hiersimu(formula, data, FUN, location = c("mean", "median"),
    relative = FALSE, drop.highest = FALSE, nsimul=99,
    method = "r2dtable", ...)
```

Arguments

`y`	A community matrix.
`x`	A matrix with same number of rows as in `y`, columns coding the levels of sampling hierarchy. The number of groups within the hierarchy must decrease from left to right. If `x` is missing, function performs an overall decomposition into alpha, beta and gamma diversities.
`formula`	A two sided model formula in the form `y ~ x`, where `y` is the community data matrix with samples as rows and species as column. Right hand side (`x`) must be grouping variables referring to levels of sampling hierarchy, terms from right to left will be treated as nested (first column is the lowest, last is the highest level). The formula will add a unique identifier to rows and constant for the rows to always produce estimates of row-level alpha and overall gamma diversities. You must use non-formula interface to avoid this behaviour. Interaction terms are not allowed.
`data`	A data frame where to look for variables defined in the right hand side of `formula`. If missing, variables are looked in the global environment.
`index`	Name of the diversity index, one of `"richness"` for the number of species, `"shannon"`, `"simpson"`, `"invsimpson"` of function `diversity`, `"hill1"` for Hill number 1 that is the exponent of `"shannon"`, or `"hill2"` for Hill number 2 that is a synonym of `"invsimpson"`.
`weights`	Character, `"unif"` for uniform weights, `"prop"` for weighting proportional to sample abundances to use in weighted averaging of individual alpha values within strata of a given level of the sampling hierarchy.
`relative`	Logical, if `TRUE` then alpha and beta diversity values are given relative to the value of gamma for function `adipart`.
`nsimul`	Number of permutations to use. If `nsimul = 0`, only the `FUN` argument is evaluated. It is thus possible to reuse the statistic values without a null model.
`method`	Null model method: either a name (character string) of a method defined in `make.commsim` or a `commsim` function. The default `"r2dtable"` keeps row sums and column sums fixed. See `oecosimu` for Details and Examples.
`FUN`	A function to be used by `hiersimu`. This must be fully specified, because currently other arguments cannot be passed to this function via `...`.
`location`	Character, identifies which function (mean or median) is to be used to calculate location of the samples.
`drop.highest`	Logical, to drop the highest level or not. When `FUN` evaluates only arrays with at least 2 dimensions, highest level should be dropped, or not selected at all.
`...`	Other arguments passed to functions, e.g. base of logarithm for Shannon diversity, or `method`, `thin` or `burnin` arguments for `oecosimu`.

Details

Additive diversity partitioning means that mean alpha and beta diversities add up to gamma diversity, thus beta diversity is measured in the same dimensions as alpha and gamma (Lande 1996). This additive procedure is then extended across multiple scales in a hierarchical sampling design with $i = 1, 2, 3, \ldots, m$ levels of sampling (Crist et al. 2003). Samples in lower hierarchical levels are nested within higher level units, thus from $i = 1$ to $i = m$ grain size is increasing under constant survey extent. At each level i, α_i denotes average diversity found within samples.

At the highest sampling level, the diversity components are calculated as

$$\beta_m = \gamma - \alpha_m$$

For each lower sampling level as

$$\beta_i = \alpha_{i+1} - \alpha_i$$

Then, the additive partition of diversity is

$$\gamma = \alpha_1 + \sum_{i=1}^{m} \beta_i$$

Average alpha components can be weighted uniformly (`weight="unif"`) to calculate it as simple average, or proportionally to sample abundances (`weight="prop"`) to calculate it as weighted average as follows

$$\alpha_i = \sum_{j=1}^{n_i} D_{ij} w_{ij}$$

where D_{ij} is the diversity index and w_{ij} is the weight calculated for the jth sample at the ith sampling level.

The implementation of additive diversity partitioning in `adipart` follows Crist et al. 2003. It is based on species richness (S, not $S - 1$), Shannon's and Simpson's diversity indices stated as the `index` argument.

The expected diversity components are calculated `nsimul` times by individual based randomisation of the community data matrix. This is done by the `"r2dtable"` method in `oecosimu` by default.

`hiersimu` works almost in the same way as `adipart`, but without comparing the actual statistic values returned by `FUN` to the highest possible value (cf. gamma diversity). This is so, because in most of the cases, it is difficult to ensure additive properties of the mean statistic values along the hierarchy.

Value

An object of class `"adipart"` or `"hiersimu"` with same structure as `oecosimu` objects.

Author(s)

Péter Sólymos, <solymos@ualberta.ca>

References

Crist, T.O., Veech, J.A., Gering, J.C. and Summerville, K.S. (2003). Partitioning species diversity across landscapes and regions: a hierarchical analysis of α, β, and γ-diversity. *Am. Nat.*, **162**, 734–743.

Lande, R. (1996). Statistics and partitioning of species diversity, and similarity among multiple communities. *Oikos*, **76**, 5–13.

See Also

See `oecosimu` for permutation settings and calculating p-values. `multipart` for multiplicative diversity partitioning.

Examples

```
## NOTE: 'nsimul' argument usually needs to be >= 99
## here much lower value is used for demonstration

data(mite)
data(mite.xy)
data(mite.env)
## Function to get equal area partitions of the mite data
cutter <- function (x, cut = seq(0, 10, by = 2.5)) {
    out <- rep(1, length(x))
    for (i in 2:(length(cut) - 1))
        out[which(x > cut[i] & x <= cut[(i + 1)])] <- i
    return(out)}
## The hierarchy of sample aggregation
levsm <- with(mite.xy, data.frame(
    l1=1:nrow(mite),
    l2=cutter(y, cut = seq(0, 10, by = 2.5)),
    l3=cutter(y, cut = seq(0, 10, by = 5)),
    l4=rep(1, nrow(mite))))
## Let's see in a map
par(mfrow=c(1,3))
plot(mite.xy, main="l1", col=as.numeric(levsm$l1)+1, asp = 1)
plot(mite.xy, main="l2", col=as.numeric(levsm$l2)+1, asp = 1)
plot(mite.xy, main="l3", col=as.numeric(levsm$l3)+1, asp = 1)
par(mfrow=c(1,1))
## Additive diversity partitioning
adipart(mite, index="richness", nsimul=19)
## the next two define identical models
adipart(mite, levsm, index="richness", nsimul=19)
adipart(mite ~ l2 + l3, levsm, index="richness", nsimul=19)
## Hierarchical null model testing
## diversity analysis (similar to adipart)
hiersimu(mite, FUN=diversity, relative=TRUE, nsimul=19)
hiersimu(mite ~ l2 + l3, levsm, FUN=diversity, relative=TRUE, nsimul=19)
## Hierarchical testing with the Morisita index
morfun <- function(x) dispindmorisita(x)$imst
hiersimu(mite ~., levsm, morfun, drop.highest=TRUE, nsimul=19)
```

adonis *Permutational Multivariate Analysis of Variance Using Distance Matrices*

Description

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

Usage

```
adonis2(formula, data, permutations = 999, method = "bray",
    sqrt.dist = FALSE, add = FALSE, by = NULL,
    parallel = getOption("mc.cores"), na.action = na.fail,
    strata = NULL, ...)
```

Arguments

`formula` Model formula. The function can be called only with the formula interface. Most usual features of `formula` hold, especially as defined in `cca` and `rda`. The LHS must be either a community data matrix or a dissimilarity matrix, e.g., from `vegdist` or `dist`. If the LHS is a data matrix, function `vegdist` will be used to find the dissimilarities. The RHS defines the independent variables. These can be continuous variables or factors, they can be transformed within the formula, and they can have interactions as in a typical `formula`. The RHS can have a special term `Condition` that defines variables to be "partialled out" before constraints, just like in `dbrda` or `cca`. This allows the use of partial 'adonis2'.

`data` the data frame for the independent variables, with rows in the same order as the community data matrix or dissimilarity matrix named on the LHS of `formula`.

`permutations` a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.

`method` the name of any method used in `vegdist` to calculate pairwise distances if the left hand side of the `formula` was a data frame or a matrix.

`sqrt.dist` Take square root of dissimilarities. This often euclidifies dissimilarities.

`add` Add a constant to the non-diagonal dissimilarities such that all eigenvalues are non-negative in the underlying Principal Co-ordinates Analysis (see `wcmdscale` for details). Choice `"lingoes"` (or `TRUE`) use the recommended method of Legendre & Anderson (1999: "method 1") and `"cailliez"` uses their "method 2".

`by` `by = NULL` will assess the overall significance of all terms together, `by = "terms"` will assess significance for each term (sequentially from first to last), setting `by = "margin"` will assess the marginal effects of the terms (each marginal term analysed in a model with all other variables), `by = "onedf"` will analyse one-degree-of-freedom contrasts sequentially. The argument is passed on to `anova.cca`.

`parallel`	Number of parallel processes or a predefined socket cluster. With `parallel = 1` uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
`na.action`	Handling of missing values on the right-hand-side of the formula (see `na.fail` for explanation and alternatives). Missing values are not allowed on the left-hand-side. NB, argument `subset` is not implemented.
`strata`	Groups within which to constrain permutations. The traditional non-movable strata are set as Blocks in the **permute** package, but some more flexible alternatives may be more appropriate.
`...`	Other arguments passed to `vegdist`.

Details

`adonis2` is a function for the analysis and partitioning sums of squares using dissimilarities. The function is based on the principles of McArdle & Anderson (2001) and can perform sequential, marginal and overall tests. The function also allows using additive constants or squareroot of dissimilarities to avoid negative eigenvalues, but can also handle semimetric indices (such as Bray-Curtis) that produce negative eigenvalues. The `adonis2` tests are identical to `anova.cca` of `dbrda`. With Euclidean distances, the tests are also identical to `anova.cca` of `rda`.

The function partitions sums of squares of a multivariate data set, and they are directly analogous to MANOVA (multivariate analysis of variance). McArdle and Anderson (2001) and Anderson (2001) refer to the method as "permutational MANOVA" (formerly "nonparametric MANOVA"). Further, as the inputs are linear predictors, and a response matrix of an arbitrary number of columns, they are a robust alternative to both parametric MANOVA and to ordination methods for describing how variation is attributed to different experimental treatments or uncontrolled covariates. The method is also analogous to distance-based redundancy analysis and algorithmically similar to `dbrda` (Legendre and Anderson 1999), and provides an alternative to AMOVA (nested analysis of molecular variance, Excoffier, Smouse, and Quattro, 1992; `amova` in the **ade4** package) for both crossed and nested factors.

Value

The function returns an `anova.cca` result object with a new column for partial R^2: This is the proportion of sum of squares from the total, and in marginal models (`by = "margin"`) the R^2 terms do not add up to 1.

Note

Anderson (2001, Fig. 4) warns that the method may confound location and dispersion effects: significant differences may be caused by different within-group variation (dispersion) instead of different mean values of the groups (see Warton et al. 2012 for a general analysis). However, it seems that `adonis2` is less sensitive to dispersion effects than some of its alternatives (`anosim`, `mrpp`). Function `betadisper` is a sister function to `adonis2` to study the differences in dispersion within the same geometric framework.

Author(s)

Martin Henry H. Stevens and Jari Oksanen.

References

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

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

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

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

Warton, D.I., Wright, T.W., Wang, Y. 2012. Distance-based multivariate analyses confound location and dispersion effects. *Methods in Ecology and Evolution*, 3, 89–101.

See Also

mrpp, anosim, mantel, varpart.

Examples

```
data(dune)
data(dune.env)
## default is overall (omnibus) test
adonis2(dune ~ Management*A1, data = dune.env)
## sequential tests
adonis2(dune ~ Management*A1, data = dune.env, by = "terms")

### Example of use with strata, for nested (e.g., block) designs.
dat <- expand.grid(rep=gl(2,1), NO3=factor(c(0,10)),field=gl(3,1) )
dat
Agropyron <- with(dat, as.numeric(field) + as.numeric(NO3)+2) +rnorm(12)/2
Schizachyrium <- with(dat, as.numeric(field) - as.numeric(NO3)+2) +rnorm(12)/2
total <- Agropyron + Schizachyrium
Y <- data.frame(Agropyron, Schizachyrium)
mod <- metaMDS(Y, trace = FALSE)
plot(mod)
### Ellipsoid hulls show treatment
with(dat, ordiellipse(mod, NO3, kind = "ehull", label = TRUE))
### Spider shows fields
with(dat, ordispider(mod, field, lty=3, col="red", label = TRUE))

### Incorrect (no strata)
adonis2(Y ~ NO3, data = dat, permutations = 199)
## Correct with strata
with(dat, adonis2(Y ~ NO3, data = dat, permutations = 199, strata = field))
```

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(x, grouping, permutations = 999, distance = "bray", strata = NULL,
    parallel = getOption("mc.cores"))
```

Arguments

x	Data matrix or data frame in which rows are samples and columns are response variable(s), or a dissimilarity object or a symmetric square matrix of dissimilarities.
grouping	Factor for grouping observations.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
distance	Choice of distance metric that measures the dissimilarity between two observations. See vegdist for options. This will be used if x was not a dissimilarity structure or a symmetric square matrix.
strata	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.

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 (monoMDS), 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 $-1 \cdots +1$, 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. See permutations for additional details on permutation tests in Vegan. The distribution of simulated values can be inspected with the permustats function.

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 returns 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. The distribution of permutation values can be inspected with function permustats.
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.
control	A list of control values for the permutations as returned by the function how.

Note

The anosim function can confound the differences between groups and dispersion within groups and the results can be difficult to interpret (cf. Warton et al. 2012). The function returns a lot of information to ease studying its performance. Most anosim models could be analysed with adonis2 which seems to be a more robust alternative.

Author(s)

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

References

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

Warton, D.I., Wright, T.W., Wang, Y. 2012. Distance-based multivariate analyses confound location and dispersion effects. *Methods in Ecology and Evolution*, 3, 89–101

See Also

mrpp for a similar function using original dissimilarities instead of their ranks. dist and vegdist for obtaining dissimilarities, and rank for ranking real values. For comparing dissimilarities against continuous variables, see mantel. Function adonis2 is a more robust alternative that should preferred.

Examples

```
data(dune)
data(dune.env)
dune.dist <- vegdist(dune)
dune.ano <- with(dune.env, 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 distance-based Redundancy Analysis (dbRDA, dbrda) to assess the significance of constraints.

Usage

```
## S3 method for class 'cca'
anova(object, ..., permutations = how(nperm=999),
     by = NULL, model = c("reduced", "direct", "full"),
     parallel = getOption("mc.cores"), strata = NULL,
     cutoff = 1, scope = NULL, test = c("permutation", "F"))
## S3 method for class 'cca'
permutest(x, permutations = how(nperm = 99),
     model = c("reduced", "direct", "full"), by = NULL, first = FALSE,
     strata = NULL, parallel = getOption("mc.cores"), ...)
```

Arguments

object	One or several result objects from cca, rda, dbrda or capscale. For a single object, a test specified in by or an overall test is given. If there are several result objects, they should be nested, and they are compared against previous model in the order of nestedness.
x	A single ordination result object.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
by	The default (NULL) performs overall test for the model. In permutest and anova you can set by = "term" for sequential test of terms (from first to last), and by = "onedf" for sequential test of one-degree-of-freedom contrasts of factor levels. In anova you can also set by = "margin" for marginal effecs of terms (each marginal term analysed in a model with all other variables), and by = "axis" for each constrained axis. Choices "terms" and "margin" can be used only with models fitted with formula. by cannot be used together with first = TRUE.

model	Permutation model: model="direct" permutes community data, model="reduced" permutes residuals of the community data after Conditions (partial model), model = "full" permutes residuals after Conditions and Constraints.
parallel	Use parallel processing with the given number of cores.
strata	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata. It is an error to use this when permutations is a matrix, or a how defines blocks. This is a legacy argument: it is better to use permutations = how(..., blocks) instead.
cutoff	Only effective with by="axis" where stops permutations after an axis equals or exceeds the cutoff p-value.
scope	Only effective with by="margin" where it can be used to select the marginal terms for testing. The default is to test all marginal terms in drop.scope.
test	Test type, either permutation test or parametric F-test. Parametric test is only available for rda. It is similar as anova.mlm test = "Spherical" with Greenhouse-Geisser correction for asphericity.
first	Analyse only significance of the first axis.
...	Parameters passed to other functions. anova.cca passes all arguments to permutest.cca. In anova with by = "axis" you can use argument cutoff (defaults 1) which stops permutations after exceeding the given level.

Details

Functions anova.cca and permutest.cca implement ANOVA like permutation tests for the joint effect of constraints in cca, rda, dbrda or capscale. Function anova is intended as a more user-friendly alternative to permutest (that is the real workhorse).

Function anova can analyse a sequence of constrained ordination models. The analysis is based on the differences in residual deviance in permutations of nested models.

The default test is for the sum of all constrained eigenvalues. Setting first = TRUE will perform a test for the first constrained eigenvalue. Argument first can be set either in anova.cca or in permutest.cca. It is also possible to perform significance tests for each axis or for each term (constraining variable) using argument by in anova.cca. Setting by = "axis" will perform separate significance tests for each constrained axis (Legendre et al. 2011). You can stop permutation tests after exceeding a given significance level with argument cutoff to speed up calculations in large models. Setting by = "terms" will perform separate significance test for each term (constraining variable). The terms are assessed sequentially from first to last, and the order of the terms will influence their significances. Setting by = "onedf" will perform a similar sequential test for one-degree-of-freedom effects, where multi-level factors are split in their contrasts. Setting by = "margin" will perform separate significance test for each marginal term in a model with all other terms. The marginal test also accepts a scope argument for the drop.scope which can be a character vector of term labels that are analysed, or a fitted model of lower scope. The marginal effects are also known as "Type III" effects, but the current function only evaluates marginal terms. It will, for instance, ignore main effects that are included in interaction terms. In calculating pseudo-F, all terms are compared to the same residual of the full model.

Community data are permuted with choice model="direct", and residuals after partial CCA/ RDA/ dbRDA with choice model="reduced" (default). If there is no partial CCA/ RDA/ dbRDA stage,

model="reduced" simply permutes the data and is equivalent to model="direct". The test statistic is "pseudo-F", which is the ratio of constrained and unconstrained total Inertia (Chi-squares, variances or something similar), each divided by their respective degrees of freedom. 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/ dbRDA, the effect of conditioning variables ("covariables") is removed before permutation, and the total Chi-square is not fixed, and test based on pseudo-F would differ from the test based on plain eigenvalues.

Function can also perform parametric F-test on rda models. This test is similar to test = "Spherical" in anova.mlm. The test uses the tabulated F-values but multiplies the degrees of freedom with number of variables (species, columns) times correction factor epsilon, using Greenhouse-Geisser correction for asphericity. Preliminary tests indicate that this test is often consistent with permutation tests in rda (but not in cca and cannot be used with distance-based methods). However, if you use this test, it is best to check that it gives consistent results with permutation tests (see Examples).

Value

The function anova.cca calls permutest.cca and fills an anova table. Additional attributes are Random.seed (the random seeds used), control (the permutation design, see how) and F.perm (the permuted test statistics).

Author(s)

Jari Oksanen

References

Legendre, P. and Legendre, L. (2012). *Numerical Ecology*. 3rd English ed. Elsevier.

Legendre, P., Oksanen, J. and ter Braak, C.J.F. (2011). Testing the significance of canonical axes in redundancy analysis. *Methods in Ecology and Evolution* 2, 269–277.

See Also

cca, rda, dbrda to get something to analyse. Function adonis2 is similar to anova of a dbrda result. Function drop1.cca calls anova.cca with by = "margin", and add1.cca an analysis for single terms additions, which can be used in automatic or semiautomatic model building (see ordistep). Function permustats extracts permutation results for further analysis.

Examples

```
data(dune, dune.env)
mod <- cca(dune ~ Moisture + Management, dune.env)

## overall test
anova(mod)

## tests for individual terms and contrasts
anova(mod, by = "term")
anova(mod, by = "margin")
anova(mod, by = "onedf")
```

```
## compare several models in succession
mod0 <- cca(dune ~ 1, dune.env)
mod1 <- cca(dune ~ Moisture + A1, dune.env)
mod2 <- update(mod1, . ~ . + Management + Manure, dune.env)
anova(mod0, mod1, mod2)

## See that permutation and F-distribution are consistent
mod <- rda(dune ~ Moisture + Management, dune.env)
(ano <- anova(mod))
anova(mod, test = "F")
## adjustment for degrees of freedom in F distribution
adj <- ncol(dune) * 0.245 # Greenhouse-Geisser epsilon
qqnorm(permustats(ano), x = qf(ppoints(999), 6*adj, 13*adj, lower.tail=FALSE),
  observed = FALSE)
abline(0, 1, col = 2) # line of equality
```

---

avgdist *Averaged Subsampled Dissimilarity Matrices*

Description

The function computes the dissimilarity matrix of a dataset multiple times using vegdist while randomly subsampling the dataset each time. All of the subsampled iterations are then averaged (mean) to provide a distance matrix that represents the average of multiple subsampling iterations. This emulates the behavior of the distance matrix calculator within the Mothur microbial ecology toolkit.

Usage

```
avgdist(x, sample, distfun = vegdist, meanfun = mean,
    transf = NULL, iterations = 100, dmethod = "bray",
    diag = TRUE, upper = TRUE, ...)
```

Arguments

x	Community data matrix.
sample	The subsampling depth to be used in each iteration. Samples that do not meet this threshold will be removed from the analysis, and their identity returned to the user in stdout.
distfun	The dissimilarity matrix function to be used. Default is the vegan vegdist
meanfun	The calculation to use for the average (mean or median).
transf	Option for transforming the count data before calculating the distance matrix. Any base transformation option can be used (e.g. sqrt)
iterations	The number of random iterations to perform before averaging. Default is 100 iterations.
dmethod	Dissimilarity index to be used with the specified dissimilarity matrix function. Default is Bray-Curtis

diag, upper — Return dissimilarities with diagonal and upper triangle. NB. the default differs from vegdist and returns symmetric "dist" structure instead of lower diagonal. However, the object cannot be accessed with matrix indices unless cast to matrix with as.matrix.

... — Any additional arguments to add to the distance function or mean/median function specified.

Note

The function builds on the function rrarefy and and additional distance matrix function (e.g. vegdist) to add more meaningful representations of distances among randomly subsampled datasets by presenting the average of multiple random iterations. This function runs using the vegdist. This functionality has been utilized in the Mothur standalone microbial ecology toolkit, see https://mothur.org/wiki/Dist.shared.

Author(s)

Geoffrey Hannigan, with some minor tweaks by Gavin L. Simpson.

See Also

This function utilizes the vegdist and rrarefy functions.

Examples

```
# Import an example count dataset
data(BCI)
# Test the base functionality
mean.avg.dist <- avgdist(BCI, sample = 50, iterations = 10)
# Test the transformation function
mean.avg.dist.t <- avgdist(BCI, sample = 50, iterations = 10, transf = sqrt)
# Test the median functionality
median.avg.dist <- avgdist(BCI, sample = 50, iterations = 10, meanfun = median)
# Print the resulting tables
head(as.matrix(mean.avg.dist))
head(as.matrix(mean.avg.dist.t))
head(as.matrix(median.avg.dist))
# Run example to illustrate low variance of mean, median, and stdev results
# Mean and median std dev are around 0.05
sdd <- avgdist(BCI, sample = 50, iterations = 100, meanfun = sd)
summary(mean.avg.dist)
summary(median.avg.dist)
summary(sdd)
# Test for when subsampling depth excludes some samples
# Return samples that are removed for not meeting depth filter
depth.avg.dist <- avgdist(BCI, sample = 450, iterations = 10)
# Print the result
depth.avg.dist
```

BCI *Barro Colorado Island Tree Counts*

Description

Tree counts in 1-hectare plots in the Barro Colorado Island and associated site information.

Usage

```
data(BCI)
data(BCI.env)
```

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. The names were updated with The Plant List web service (now phased out) and Kress et al. (2009) which allows matching 207 of species against doi:10.5061/dryad.63q27 (Zanne et al., 2014). The original species names are available as attribute `original.names` of `BCI`. See Examples for changed names.

For `BCI.env`, a data frame with 50 plots (rows) and nine site variables derived from Pyke et al. (2001) and Harms et al. (2001):

`UTM.EW`: UTM coordinates (zone 17N) East-West.

`UTM.NS`: UTM coordinates (zone 17N) North-South.

`Precipitation`: Precipitation in mm per year.

`Elevation`: Elevation in m above sea level.

`Age.cat`: Forest age category.

`Geology`: The Underlying geological formation.

`Habitat`: Dominant habitat type based on the map of habitat types in 25 grid cells in each plot (Harms et al. 2001, excluding streamside habitat). The habitat types are `Young` forests (*ca.* 100 years), old forests on > 7 degree slopes (`OldSlope`), old forests under 152 m elevation (`OldLow`) and at higher elevation (`OldHigh`) and `Swamp` forests.

`River`: `"Yes"` if there is streamside habitat in the plot.

`EnvHet`: Environmental Heterogeneity assessed as the Simpson diversity of frequencies of `Habitat` types in 25 grid cells in the plot.

Details

Data give the numbers of trees at least 10 cm in diameter at breast height (DBH) in each one hectare quadrat in the 1982 BCI plot. Within each plot, all individuals were tallied and are recorded in this table. The full survey included smaller trees with DBH 1 cm or larger, but the `BCI` dataset is a subset of larger trees as compiled by Condit et al. (2002). The full data with thinner trees has densities above 4000 stems per hectare, or about ten times more stems than these data. The dataset `BCI` was provided (in 2003) to illustrate analysis methods in **vegan**. For scientific research on ecological

issues we strongly recommend to access complete and more modern data (Condit et al. 2019) with updated taxonomy (Condit et al. 2020).

The data frame contains only the Barro Colorado Island subset of the full data table of Condit et al. (2002).

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

A full description of the site information in `BCI.env` is given in Pyke et al. (2001) and Harms et al. (2001). *N.B.* Pyke et al. (2001) and Harms et al. (2001) give conflicting information about forest age categories and elevation.

Source

https://www.science.org/doi/10.1126/science.1066854 for community data and References for environmental data. For updated complete data (incl. thinner trees down to 1 cm), see Condit et al. (2019).

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.

Condit R., Pérez, R., Aguilar, S., Lao, S., Foster, R. & Hubbell, S. (2019). Complete data from the Barro Colorado 50-ha plot: 423617 trees, 35 years [Dataset]. *Dryad*. doi:10.15146/5xcp0d46

Condit, R., Aguilar, S., Lao, S., Foster, R., Hubbell, S. (2020). BCI 50-ha Plot Taxonomy [Dataset]. *Dryad*. doi:10.15146/R3FH61

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

Kress W.J., Erickson D.L, Jones F.A., Swenson N.G, Perez R., Sanjur O. & Bermingham E. (2009) Plant DNA barcodes and a community phylogeny of a tropical forest dynamics plot in Panama. *PNAS* 106, 18621–18626.

Pyke, C. R., Condit, R., Aguilar, S., & Lao, S. (2001). Floristic composition across a climatic gradient in a neotropical lowland forest. *Journal of Vegetation Science* 12, 553–566. doi:10.2307/3237007

Zanne A.E., Tank D.C., Cornwell, W.K., Eastman J.M., Smith, S.A., FitzJohn, R.G., McGlinn, D.J., O'Meara, B.C., Moles, A.T., Reich, P.B., Royer, D.L., Soltis, D.E., Stevens, P.F., Westoby, M., Wright, I.J., Aarssen, L., Bertin, R.I., Calaminus, A., Govaerts, R., Hemmings, F., Leishman, M.R., Oleksyn, J., Soltis, P.S., Swenson, N.G., Warman, L. & Beaulieu, J.M. (2014) Three keys to the radiation of angiosperms into freezing environments. *Nature* 506, 89–92. doi:10.1038/nature12872 (published online Dec 22, 2013).

See Also

Extra-CRAN package **natto** (https://github.com/jarioksa/natto) has data set `BCI.env2` with original grid data of Harms et al. (2001) habitat classification, and data set `BCI.taxon` of APG III classification of tree species.

Examples

```
data(BCI, BCI.env)
head(BCI.env)
## see changed species names
oldnames <- attr(BCI, "original.names")
taxa <- cbind("Old Names" = oldnames, "Current Names" = names(BCI))
noquote(taxa[taxa[,1] != taxa[,2], ])
```

`beals` *Beals Smoothing and Degree of Absence*

Description

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

Usage

```
beals(x, species = NA, reference = x, type = 0, include = TRUE)
swan(x, maxit = Inf, type = 0)
```

Arguments

`x`	Community data frame or matrix.
`species`	Column index used to compute Beals function for a single species. The default (`NA`) indicates that the function will be computed for all species.
`reference`	Community data frame or matrix to be used to compute joint occurrences. By default, `x` is used as reference to compute the joint occurrences.
`type`	Numeric. Specifies if and how abundance values have to be used in function `beals`. See details for more explanation.
`include`	This logical flag indicates whether the target species has to be included when computing the mean of the conditioned probabilities. The original Beals (1984) definition is equivalent to `include=TRUE`, while the formulation of Münzbergová and Herben is equal to `include=FALSE`.
`maxit`	Maximum number of iterations. The default `Inf` means that iterations are continued until there are no zeros or the number of zeros does not change. Probably only `maxit = 1` makes sense in addition to the default.

Details

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

interpreted as a mean of conditional probability, the `beals` function can be interpreted as a mean of conditioned probabilities (De Cáceres & Legendre 2008). The present function is generalized to abundance values (De Cáceres & Legendre 2008).

The `type` argument specifies if and how abundance values have to be used. `type = 0` presence/absence mode. `type = 1` abundances in `reference` (or `x`) are used to compute conditioned probabilities. `type = 2` abundances in `x` are used to compute weighted averages of conditioned probabilities. `type = 3` abundances are used to compute both conditioned probabilities and weighted averages.

Beals smoothing was originally suggested as a method of data transformation to remove excessive zeros (Beals 1984, McCune 1994). However, it is not a suitable method for this purpose since it does not maintain the information on species presences: a species may have a higher probability of occurrence at a site where it does not occur than at sites where it occurs. Moreover, it regularizes data too strongly. The method may be useful in identifying species that belong to the species pool (Ewald 2002) or to identify suitable unoccupied patches in metapopulation analysis (Münzbergová & Herben 2004). In this case, the function should be called with `include=FALSE` for cross-validation smoothing for species; argument `species` can be used if only one species is studied.

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

Value

The function returns a transformed data matrix or a vector if Beals smoothing is requested for a single species.

Author(s)

Miquel De Cáceres and Jari Oksanen

References

Beals, E.W. 1984. Bray-Curtis ordination: an effective strategy for analysis of multivariate ecological data. Pp. 1–55 in: MacFadyen, A. & E.D. Ford [eds.] *Advances in Ecological Research, 14*. Academic Press, London.

De Cáceres, M. & Legendre, P. 2008. Beals smoothing revisited. *Oecologia* 156: 657–669.

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

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

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

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

See Also

decostand for proper standardization methods, specpool for an attempt to assess the size of species pool. Function indpower assesses the power of each species to estimate the probabilities predicted by beals.

Examples

```
data(dune)
## Default
x <- beals(dune)
## Remove target species
x <- beals(dune, include = FALSE)
## Smoothed values against presence or absence of species
pa <- decostand(dune, "pa")
boxplot(as.vector(x) ~ unlist(pa), xlab="Presence", ylab="Beals")
## Remove the bias of tarbet species: Yields lower values.
beals(dune, type =3, include = FALSE)
## Uses abundance information.
## Vector with beals smoothing values corresponding to the first species
## in dune.
beals(dune, species=1, include=TRUE)
```

---

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

Description

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

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

Usage

```
betadisper(d, group, type = c("median","centroid"), bias.adjust = FALSE,
       sqrt.dist = FALSE, add = FALSE)

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

## S3 method for class 'betadisper'
```

```
scores(x, display = c("sites", "centroids"),
       choices = c(1,2), ...)

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

## S3 method for class 'betadisper'
plot(x, axes = c(1,2), cex = 0.7,
     pch = seq_len(ng), col = NULL, lty = "solid", lwd = 1, hull = TRUE,
     ellipse = FALSE, conf,
     segments = TRUE, seg.col = "grey", seg.lty = lty, seg.lwd = lwd,
     label = TRUE, label.cex = 1,
     ylab, xlab, main, sub, ...)

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

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

## S3 method for class 'betadisper'
print(x, digits = max(3, getOption("digits") - 3),
                           neigen = 8, ...)

betadistances(x, ...)
```

Arguments

d	a distance structure such as that returned by dist, betadiver or vegdist.
group	vector describing the group structure, usually a factor or an object that can be coerced to a factor using as.factor. Can consist of a factor with a single level (i.e., one group).
type	the type of analysis to perform. Use the spatial median or the group centroid? The spatial median is now the default.
bias.adjust	logical: adjust for small sample bias in beta diversity estimates?
sqrt.dist	Take square root of dissimilarities. This often euclidifies dissimilarities.
add	Add a constant to the non-diagonal dissimilarities such that all eigenvalues are non-negative in the underlying Principal Co-ordinates Analysis (see wcmdscale for details). Choice "lingoes" (or TRUE) use the recommended method of Legendre & Anderson (1999: "method 1") and "cailliez" uses their "method 2".
display	character; partial match to access scores for "sites" or "species".
object, x	an object of class "betadisper", the result of a call to betadisper.
choices, axes	the principal coordinate axes wanted.
hull	logical; should the convex hull for each group be plotted?
ellipse	logical; should the standard deviation data ellipse for each group be plotted?

`conf`	Expected fractions of data coverage for data ellipses, e.g. 0.95. The default is to draw a 1 standard deviation data ellipse, but if supplied, `conf` is multiplied with the corresponding value found from the Chi-squared distribution with 2df to provide the requested coverage (probability contour).
`pch`	plot symbols for the groups, a vector of length equal to the number of groups.
`col`	colors for the plot symbols and centroid labels for the groups, a vector of length equal to the number of groups.
`lty`, `lwd`	linetype, linewidth for convex hulls and confidence ellipses.
`segments`	logical; should segments joining points to their centroid be drawn?
`seg.col`	colour to draw segments between points and their centroid. Can be a vector, in which case one colour per group.
`seg.lty`, `seg.lwd`	linetype and line width for segments.
`label`	logical; should the centroids by labelled with their respective factor label?
`label.cex`	numeric; character expansion for centroid labels.
`cex`, `ylab`, `xlab`, `main`, `sub`	graphical parameters. For details, see `plot.default`.
`which`	A character vector listing terms in the fitted model for which the intervals should be calculated. Defaults to the grouping factor.
`ordered`	logical; see `TukeyHSD`.
`conf.level`	A numeric value between zero and one giving the family-wise confidence level to use.
`digits`, `neigen`	numeric; for the `print` method, sets the number of digits to use (as per `print.default`) and the maximum number of axes to display eigenvalues for, respectively.
`...`	arguments, including graphical parameters (for `plot.betadisper` and `boxplot.betadisper`), passed to other methods.

Details

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

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

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

The distance to its centroid of a point is

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

where Δ^2 is the squared Euclidean distance between u_{ij}, the principal coordinate for the jth point in the ith group, and c_i, the coordinate of the centroid for the ith group. The super-scripted '$+$' and '$-$' indicate the real and imaginary parts respectively. This is equation (3) in Anderson (2006). If the imaginary part is greater in magnitude than the real part, then we would be taking the square root of a negative value, resulting in NaN, and these cases are changed to zero distances (with a warning). This is in line with the behaviour of Marti Anderson's PERMDISP2 programme. Function `betadistances` returns distances from all points to all centroids. Moreover, it gives the original `group` and nearest group for each point.

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

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

The results of the analysis can be visualised using the `plot` and `boxplot` methods. The distances of points to all centroids (`group`) can be found with function `betadistances`.

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

As noted in passing by Anderson (2006) and in a related context by O'Neill (2000), estimates of dispersion around a central location (median or centroid) that is calculated from the same data will be biased downward. This bias matters most when comparing diversity among treatments with small, unequal numbers of samples. Setting `bias.adjust=TRUE` when using `betadisper` imposes a $\sqrt{n/(n-1)}$ correction (Stier et al. 2013).

Value

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

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

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

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

`eigenvals.betadisper` returns a named vector of eigenvalues.

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

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

`eig`	numeric; the eigenvalues of the principal coordinates analysis.
`vectors`	matrix; the eigenvectors of the principal coordinates analysis.

`distances`	numeric; the Euclidean distances in principal coordinate space between the samples and their respective group centroid or median.
`group`	factor; vector describing the group structure
`centroids`	matrix; the locations of the group centroids or medians on the principal coordinates.
`group.distances`	numeric; the mean distance to each group centroid or median.
`call`	the matched function call.

Warning

Stewart Schultz noticed that the permutation test for `type="centroid"` had the wrong type I error and was anti-conservative. As such, the default for `type` has been changed to `"median"`, which uses the spatial median as the group centroid. Tests suggests that the permutation test for this type of analysis gives the correct error rates.

Note

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

Missing values in either `d` or `group` will be removed prior to performing the analysis.

Author(s)

Gavin L. Simpson; bias correction by Adrian Stier and Ben Bolker.

References

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

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

O'Neill, M.E. (2000) A Weighted Least Squares Approach to Levene's Test of Homogeneity of Variance. *Australian & New Zealand Journal of Statistics* **42**, 81-–100.

Stier, A.C., Geange, S.W., Hanson, K.M., & Bolker, B.M. (2013) Predator density and timing of arrival affect reef fish community assembly. *Ecology* **94**, 1057–1068.

See Also

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

Examples

```
data(varespec)

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

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

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

## Perform test
anova(mod)

## Permutation test for F
permutest(mod, pairwise = TRUE, permutations = 99)

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

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

## with data ellipses instead of hulls
plot(mod, ellipse = TRUE, hull = FALSE) # 1 sd data ellipse
plot(mod, ellipse = TRUE, hull = FALSE, conf = 0.90) # 90% data ellipse

# plot with manual colour specification
my_cols <- c("#1b9e77", "#7570b3")
plot(mod, col = my_cols, pch = c(16,17), cex = 1.1)

## can also specify which axes to plot, ordering respected
plot(mod, axes = c(3,1), seg.col = "forestgreen", seg.lty = "dashed")

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

## `scores` and `eigenvals` also work
scrs <- scores(mod)
str(scrs)
head(scores(mod, 1:4, display = "sites"))
# group centroids/medians
scores(mod, 1:4, display = "centroids")
# eigenvalues from the underlying principal coordinates analysis
eigenvals(mod)

## try out bias correction; compare with mod3
(mod3B <- betadisper(dis, groups, type = "median", bias.adjust=TRUE))
anova(mod3B)
permutest(mod3B, permutations = 99)

## should always work for a single group
group <- factor(rep("grazed", NROW(varespec)))
```

```
(tmp <- betadisper(dis, group, type = "median"))
(tmp <- betadisper(dis, group, type = "centroid"))

## simulate missing values in 'd' and 'group'
## using spatial medians
groups[c(2,20)] <- NA
dis[c(2, 20)] <- NA
mod2 <- betadisper(dis, groups) ## messages
mod2
permutest(mod2, permutations = 99)
anova(mod2)
plot(mod2)
boxplot(mod2)
plot(TukeyHSD(mod2))

## Using group centroids
mod3 <- betadisper(dis, groups, type = "centroid")
mod3
permutest(mod3, permutations = 99)
anova(mod3)
plot(mod3)
boxplot(mod3)
plot(TukeyHSD(mod3))
```

`betadiver` *Indices of beta Diversity*

Description

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

Usage

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

Arguments

`x`	Community data matrix, or the `betadiver` result for `plot` and `scores` functions.
`method`	The index of beta diversity as defined in Koleff et al. (2003), Table 1. You can use either the subscript of β or the number of the index. See argument `help` below.
`order`	Order sites by increasing number of species. This will influence the configuration in the triangular plot and non-symmetric indices.

`help`	Show the numbers, subscript names and the defining equations of the indices and exit.
`triangular`	Return scores suitable for triangular plotting of proportions. If `FALSE`, returns a 3-column matrix of raw counts.
`...`	Other arguments to functions.

Details

The most commonly used index of beta diversity is $\beta_w = S/\alpha - 1$, where S is the total number of species, and α is the average number of species per site (Whittaker 1960). A drawback of this model is that S increases with sample size, but the expectation of α remains constant, and so the beta diversity increases with sample size. A solution to this problem is to study the beta diversity of pairs of sites (Marion et al. 2017). If we denote the number of species shared between two sites as a and the numbers of unique species (not shared) as b and c, then $S = a+b+c$ and $\alpha = (2a+b+c)/2$ so that $\beta_w = (b+c)/(2a+b+c)$. This is the Sørensen dissimilarity as defined in **vegan** function `vegdist` with argument `binary = TRUE`. Many other indices are dissimilarity indices as well.

Function `betadiver` finds all indices reviewed by Koleff et al. (2003). All these indices could be found with function `designdist`, but the current function provides a conventional shortcut. The function only finds the indices. The proper analysis must be done with functions such as `betadisper`, `adonis2` or `mantel`.

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

Value

With `method = NA`, the function returns an object of class `"betadisper"` with elements `a`, `b`, and `c`. If `method` is specified, the function returns a `"dist"` object which can be used in any function analysing dissimilarities. For beta diversity, particularly useful functions are `betadisper` to study the betadiversity in groups, `adonis2` for any model, and `mantel` to compare beta diversities to other dissimilarities or distances (including geographical distances). Although `betadiver` returns a `"dist"` object, some indices are similarities and cannot be used as such in place of dissimilarities, but that is a user error. Functions 10 (`"j"`), 11 (`"sor"`) and 21 (`"rlb"`) are similarity indices. Function sets argument `"maxdist"` similarly as `vegdist`, using `NA` when there is no fixed upper limit, and 0 for similarities.

Warning

Some indices return similarities instead of dissimilarities.

Author(s)

Jari Oksanen

References

Baselga, A. (2010) Partitioning the turnover and nestedness components of beta diversity. *Global Ecology and Biogeography* 19, 134–143.

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

Marion, Z.H., Fordyce, J.A. and Fitzpatrick, B.M. (2017) Pairwise beta diversity resolves an under-appreciated source of confusion in calculating species turnover. *Ecology* 98, 933–939.

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

See Also

`designdist` can be used to implement all these functions, and also allows using notation with `alpha` and `gamma` diversities. `vegdist` has some canned alternatives. Functions `betadisper`, `adonis2` and `mantel` can be used for analysing beta diversity objects. The returned dissimilarities can be used in any distance-based methods, such as `metaMDS`, `capscale` and `dbrda`. Functions `nestedbetasor` and `nestedbetajac` implement decomposition beta diversity measures (Sørensen and Jaccard) into turnover and nestedness components following Baselga (2010).

Examples

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

---

bgdispersal *Coefficients of Biogeographical Dispersal Direction*

Description

This function computes coefficients of dispersal direction between geographically connected areas, as defined by Legendre and Legendre (1984), and also described in Legendre and Legendre (2012, section 13.3.4).

Usage

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

Arguments

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

Details

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

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

Value

Function bgdispersal returns a list containing the following matrices:

DD1	$DD1_{j,k} = (a(b-c))/((a+b+c)^2)$
DD2	$DD2_{j,k} = (2a(b-c))/((2a+b+c)(a+b+c))$ where a, b, and c have the same meaning as in the computation of binary similarity coefficients.
DD3	$DD3_{j,k} = W(A-B)/(A+B-W)^2$
DD4	$DD4_{j,k} = 2W(A-B)/((A+B)(A+B-W))$ where W = sum(pmin(vector1, vector2)), A = sum(vector1), B = sum(vector2)
McNemar	McNemar chi-square statistic of asymmetry (Sokal and Rohlf 1995): $2(b\log(b)+c\log(c)-(b+c)\log((b+c)/2))/q$, where $q = 1+1/(2(b+c))$ (Williams correction for continuity)
prob.McNemar	probabilities associated with McNemar statistics, chi-square test. H0: no asymmetry in $(b-c)$.

Note

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

Author(s)

Pierre Legendre, Departement de Sciences Biologiques, Universite de Montreal

References

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

Legendre, P. and L. Legendre. 2012. *Numerical ecology*, 3rd English edition. Elsevier Science BV, Amsterdam.

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

Examples

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

---

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

Description

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

Usage

```
## Default S3 method:
bioenv(comm, env, method = "spearman", index = "bray",
       upto = ncol(env), trace = FALSE, partial = NULL,
       metric = c("euclidean", "mahalanobis", "manhattan", "gower"),
       parallel = getOption("mc.cores"), ...)
## S3 method for class 'formula'
bioenv(formula, data, ...)
bioenvdist(x, which = "best")
```

Arguments

comm	Community data frame or a dissimilarity object or a square matrix that can be interpreted as dissimilarities.
env	Data frame of continuous environmental variables.
method	The correlation method used in cor.

index	The dissimilarity index used for community data (comm) in vegdist. This is ignored if comm are dissimilarities.
upto	Maximum number of parameters in studied subsets.
formula, data	Model formula and data.
trace	Trace the calculations
partial	Dissimilarities partialled out when inspecting variables in env.
metric	Metric used for distances of environmental distances. See Details.
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
x	bioenv result object.
which	The number of the model for which the environmental distances are evaluated, or the "best" model.
...	Other arguments passed to vegdist.

Details

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

The argument metric defines distances in the given set of environmental variables. With metric = "euclidean", the variables are scaled to unit variance and Euclidean distances are calculated. With metric = "mahalanobis", the Mahalanobis distances are calculated: in addition to scaling to unit variance, the matrix of the current set of environmental variables is also made orthogonal (uncorrelated). With metric = "manhattan", the variables are scaled to unit range and Manhattan distances are calculated, so that the distances are sums of differences of environmental variables. With metric = "gower", the Gower distances are calculated using function daisy. This allows also using factor variables, but with continuous variables the results are equal to metric = "manhattan".

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

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

Function bioenvdist recalculates the environmental distances used within the function. The default is to calculate distances for the best model, but the number of any model can be given.

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

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

Value

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

Note

If you want to study the 'significance' of `bioenv` results, you can use function `mantel` or `mantel.partial` which use the same definition of correlation. However, `bioenv` standardizes environmental variables depending on the used metric, and you must do the same in `mantel` for comparable results (the standardized data are returned as item `x` in the result object). It is safest to use `bioenvdist` to extract the environmental distances that really were used within `bioenv`. NB., `bioenv` selects variables to maximize the Mantel correlation, and significance tests based on *a priori* selection of variables are biased.

Author(s)

Jari Oksanen

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, `monoMDS` and `metaMDS` 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
## IGNORE_RDIFF_BEGIN
summary(sol)
## IGNORE_RDIFF_END
```

biplot.rda *PCA biplot*

Description

Draws a PCA biplot with species scores indicated by biplot arrows

Usage

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

Arguments

x	A rda result object.
choices	Axes to show.
scaling	Scaling for species and site scores. Either species (2) or site (1) scores are scaled by eigenvalues, and the other set of scores is left unscaled, or with 3 both are scaled symmetrically by square root of eigenvalues. With negative scaling values in rda, species scores are divided by standard deviation of each species and multiplied with an equalizing constant. Unscaled raw scores stored in the result can be accessed with scaling = 0. The type of scores can also be specified as one of "none", "sites", "species", or "symmetric", which correspond to the values 0, 1, 2, and 3 respectively. Argument correlation can be used in combination with these character descriptions to get the corresponding negative value.
correlation	logical; if scaling is a character description of the scaling type, correlation can be used to select correlation-like scores for PCA. See argument scaling for details.
display	Scores shown. These must some of the alternatives "species" for species scores, and/or "sites" for site scores.
type	Type of plot: partial match to text for text labels, points for points, and none for setting frames only. If omitted, text is selected for smaller data sets, and points for larger. Can be of length 2 (e.g. type = c("text", "points")), in which case the first element describes how species scores are handled, and the second how site scores are drawn.
xlim, ylim	the x and y limits (min, max) of the plot.
col	Colours used for sites and species (in this order). If only one colour is given, it is used for both.
const	General scaling constant for scores.rda.
...	Other parameters passed to plot.cca.

Details

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

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

Arrow heads are at the actual species scores, and the possible text is projected to direction of the arrow.

Value

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

Note

Prior to **vegan** 2.7-2 the species text was at the actual location and arrows were shorter than with points.

Author(s)

Gavin Simpson and Jari Oksanen.

See Also

plot.cca can also draw biplot arrows since **vegan** 2.7-0.

Examples

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

## plot.cca can do the same
plot(mod, scaling = "symmetric", spe.par = list(arrows=TRUE, optimize=TRUE))

## different type for species and site scores
biplot(mod, scaling = "symmetric", type = c("text", "points"), optimize=TRUE)

## We can use ordiplot pipes to build similar plots with flexible
## control
plot(mod, scaling = "symmetric", type="n") |>
   points("sites", cex=0.7) |>
   text("species", arrows=TRUE, length=0.05, col=2, cex=0.7, font=3)
```

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

Description

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

Usage

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

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

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

Arguments

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

gridcol	The colour of the grid lines in the plots. NA, which is the default value, removes the grid lines.
...	Other parameters to the functions (ignored).
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.

Details

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

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

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

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

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

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

If sortg=TRUE, the objects are reordered by the following procedure: (1) a simple matching distance matrix is computed among the objects, based on the table of K-means assignments to groups, from K = min.g to K = max.g. (2) A principal coordinate analysis (PCoA, Gower 1966) is computed on the centred distance matrix. (3) The first principal coordinate is used as the new order of the objects in the graph.

Value

Function cascadeKM returns an object of class cascadeKM with items:

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

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

Note

The analysis can be replicated with the same random number seed only in non-parallel analysis. Runs with parallel processing cannot be replicated.

Author(s)

Marie-Helene Ouellette <Marie-Helene.Ouellette@UMontreal.ca>, Sebastien Durand <Sebastien.Durand@UMontreal. and Pierre Legendre <Pierre.Legendre@UMontreal.ca>. Parallel processing by Virgilio Gómez-Rubio. Edited for **vegan** by Jari Oksanen.

References

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

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

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

Legendre, P. & L. Legendre. 2012. *Numerical ecology*, 3rd English edition. Elsevier Science BV, Amsterdam.

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

Weingessel, A., Dimitriadou, A. and Dolnicar, S. 2002. An examination of indexes for determining the number of clusters in binary data sets. *Psychometrika* **67**: 137–160.

See Also

kmeans.

Examples

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

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

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

---

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

Description

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

Usage

```
## S3 method for class 'formula'
cca(formula, data, na.action = na.fail, subset = NULL,
  ...)
## S3 method for class 'formula'
rda(formula, data, scale=FALSE, na.action = na.fail,
  subset = NULL, ...)
## Default S3 method:
cca(X, Y, Z, ...)
## Default S3 method:
rda(X, Y, Z, scale=FALSE, ...)
ca(X, ...)
pca(X, scale=FALSE, ...)
```

Arguments

formula	Model formula, where the left hand side gives the community data matrix, right hand side gives the constraining variables, and conditioning variables can be given within a special function Condition.
data	Data frame containing the variables on the right hand side of the model formula.
X	Community data matrix.

Y	Constraining matrix, typically of environmental variables. Can be missing. If this is a data.frame, it will be expanded to a model.matrix where factors are expanded to contrasts ("dummy variables"). It is better to use formula instead of this argument, and some further analyses only work when formula was used.
Z	Conditioning matrix, the effect of which is removed ("partialled out") before next step. Can be missing. If this is a data.frame, it is expanded similarly as constraining matrix.
scale	Scale species to unit variance (like correlations).
na.action	Handling of missing values in constraints or conditions. The default (na.fail) is to stop with missing value. Choice na.omit removes all rows with missing values. Choice na.exclude keeps all observations but gives NA for results that cannot be calculated. The WA scores of rows may be found also for missing values in constraints. Missing values are never allowed in dependent community data.
subset	Subset of data rows. This can be a logical vector which is TRUE for kept observations, or a logical expression which can contain variables in the working environment, data or species names of the community data.
...	Other arguments for print or plot functions (ignored in other functions). For pca() and ca(), arguments are passed to rda() and cca(), respectively.

Details

Since their introduction (ter Braak 1986), constrained, or canonical, correspondence analysis and its spin-off, redundancy analysis, have been the most popular ordination methods in community ecology. Functions cca and rda are similar to popular proprietary software Canoco, although the implementation is completely different. The functions are based on Legendre & Legendre's (2012) 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. Legendre & Legendre (2012), Table 11.5 (p. 650) give a skeleton of the RDA algorithm of **vegan**. The algorithm of CCA is similar, but involves standardization by row and column weights.

The functions cca() and rda() can be called either with matrix-like entries for community data and constraints, or with formula interface. In general, the formula interface is preferred, because it allows a better control of the model and allows factor constraints. Some analyses of ordination results are only possible if model was fitted with formula (e.g., most cases of anova.cca, automatic model building).

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

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

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

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

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

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

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

The unconstrained ordination methods, Principal Components Analysis (PCA) and Correspondence Analysis (CA), may be performed using pca() and ca(), which are simple wrappers around rda() and cca(), respectively. Functions pca() and ca() can only be called with matrix-like objects.

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

Value

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

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

Functions pca() and ca() return objects of class "vegan_pca" and "vegan_ca" respectively to avoid clashes with other packages. These classes inherit from "rda" and "cca" respectively.

Author(s)

The responsible author was Jari Oksanen, but the code borrows heavily from Dave Roberts (Montana State University, USA).

References

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

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd 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

This help page describes two constrained ordination functions, cca and rda and their corresponding unconstrained ordination functions, ca and pca. A related method, distance-based redundancy analysis (dbRDA) is described separately (capscale), as is dbRDA's unconstrained variant, principal coordinates analysis (PCO). All these functions return similar objects (described in cca.object). There are numerous support functions that can be used to access the result object. In the list below, functions of type cca will handle all three constrained ordination objects, and functions of rda only handle rda and capscale results.

The main plotting functions are plot.cca for all methods, and biplot.rda for RDA and dbRDA. However, generic **vegan** plotting functions can also handle the results. The scores can be accessed and scaled with scores.cca, and summarized with summary.cca. The eigenvalues can be accessed with eigenvals.cca and the regression coefficients for constraints with coef.cca. The eigenvalues can be plotted with screeplot.cca, and the (adjusted) R^2 can be found with RsquareAdj.rda. The scores can be also calculated for new data sets with predict.cca which allows adding points to ordinations. The values of constraints can be inferred from ordination and community composition with calibrate.cca.

Diagnostic statistics can be found with goodness.cca, inertcomp, spenvcor, intersetcor, tolerance.cca, and vif.cca. Function as.mlm.cca refits the result object as a multiple lm object, and this allows finding influence statistics (lm.influence, cooks.distance etc.).

Permutation based significance for the overall model, single constraining variables or axes can be found with anova.cca. Automatic model building with R step function is possible with deviance.cca, add1.cca and drop1.cca. Functions ordistep and ordiR2step (for RDA) are special functions for constrained ordination. Randomized data sets can be generated with simulate.cca.

Separate methods based on constrained ordination model are principal response curves (prc) and variance partitioning between several components (varpart).

Design decisions are explained in vignette on "Design decisions" which can be accessed with browseVignettes("vegan").

Examples

```
data(varespec, 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, spe.par = list(optimize = TRUE))
## Formula interface and a better model
vare.cca <- cca(varespec ~ Al + P*(K + Baresoil), data=varechem)
vare.cca
plot(vare.cca, spe.par = list(optimize = TRUE))
## Interpretation of biplot arrows: project points to the arrow
plot(vare.cca, type = "p", bip.par = list(type = "t"))
ordisurf(vare.cca ~ Al, varechem, display = "lc", add = TRUE, knots = 1)
title(main = "Interpretation of biplot arrow (Al)")
## Partialling out and negative components of variance
cca(varespec ~ Ca, varechem)
cca(varespec ~ Ca + Condition(pH), varechem)
## RDA
data(dune, dune.env)
dune.Manure <- rda(dune ~ Manure, dune.env)
## select only best-fitting species
good <- goodness(dune.Manure, choices=1:2, summarize=TRUE)
sort(good)
plot(dune.Manure, spe.par = list(optimize=TRUE, select = good > 0.2))
```

---

cca.object *Result Object from Constrained Ordination*

Description

Ordination methods cca, rda, dbrda and capscale return similar result objects. All these methods use the same internal function ordConstrained. They differ only in (1) initial transformation of the data and in defining inertia, (2) weighting, and (3) the use of rectangular rows $\times$ columns data or symmetric rows $\times$ rows dissimilarities: rda initializes data to give variance or correlations as inertia, cca is based on double-standardized data to give Chi-square inertia and uses row and column weights, capscale maps the real part of dissimilarities to rectangular data and performs RDA, and dbrda performs an RDA-like analysis directly on symmetric dissimilarities.

Function ordConstrained returns the same result components for all these methods, and the calling function may add some more components to the final result. However, you should not access these result components directly (using $): the internal structure is not regarded as stable application interface (API), but it can change at any release. If you access the results components directly, you take a risk of breakage at any **vegan** release. The **vegan** provides a wide set of accessor functions to those components, and these functions are updated when the result object changes. This documentation gives an overview of accessor functions to the cca result object.

Usage

```
ordConstrained(Y, X = NULL, Z = NULL, method = c("cca", "rda",
    "capscale", "dbrda", "pass"), arg = FALSE)
ordiYbar(x, model = c("CCA", "CA", "pCCA", "partial", "initial"))
## S3 method for class 'cca'
model.frame(formula, ...)
## S3 method for class 'cca'
model.matrix(object, ...)
## S3 method for class 'cca'
weights(object, display = "sites", ...)
```

Arguments

Y, X, Z	Input matrix of responses (Y) and model matrices of constraints (X) and conditions (Z). NULL matrices are ignored.
method	Ordination method. If "pass", method is defined by Y of fitted ordination model.
arg	Possible extra argument passed to method; in rda this is scale.
object, x, formula	A result object from cca, rda, dbrda, or capscale.
model	Show constrained ("CCA"), unconstrained ("CA") or conditioned "partial" ("pCCA") results. In ordiYbar the value can also be "initial" for the internal working input data, and "partial" for the internal working input data after removing the partial effects.
display	Display either "sites" or "species".
...	Other arguments passed to the the function.

Details

Function ordConstrained is used as the basic analysis engine in all [partial] constrained ordination methods (cca, rda, dbrda _etc_). Those higher-level functions set up the data for ordConstrained and add items used in printing or further analysis of the results. Function ordConstrained can also be called directly, but the user should be very careful in setting up the input arguments and in processing the result. All supported ordination methods have an initXXX function that transforms the data for analysis and sets data attributes for analytic steps. These arguments include attribute METHOD and for cca row and column weights (attributes RW and CW). These are used to select the branch of analysis in further steps. Argument method selects the ordination type. method = "pass" is used when the dependent matrix Y already was processed with an initXXX method. This allows embedding ordConstrained in other functions with any constrained ordination method or using your own initXXX method (see Examples). Dependent matrix Y must be supplied but conditions Z and constraints X can be NULL and then this analysis step is skipped (if both are NULL, unconstrained ordination is performed). Steps for conditions (partialling) and constraints return residuals of input Y for the next step, and all methods return their analysis results.

The internal ("working") form of the dependent (community) data can be accessed with function ordiYbar. The form depends on the ordination method: for instance, in cca the data are weighted and Chi-square transformed, and in dbrda they are Gower-centred dissimilarities. The input data in the original ("response") form can be accessed with fitted.cca and residuals.cca. Function predict.cca can return either working or response data, and also their lower-rank approximations.

The model matrix of independent data ("Constraints" and "Conditions") can be extracted with `model.matrix`. In partial analysis, the function returns a list of design matrices called `Conditions` and `Constraints`. If either component was missing, a single matrix is returned. The redundant (aliased) terms do not appear in the model matrix. These terms can be found with `alias.cca`. Function `model.frame` tries to reconstruct the data frame from which the model matrices were derived. This is only possible if the original model was fitted with `formula` and `data` arguments, and still fails if the `data` are unavailable.

The number of observations can be accessed with `nobs.cca`, and the residual degrees of freedom with `df.residual.cca`. The information on observations with missing values can be accessed with `na.action`. The terms and formula of the fitted model can be accessed with `formula` and `terms`.

The weights used in `cca` can be accessed with `weights`. In unweighted methods (`rda`) all weights are equal.

The ordination results are saved in separate components for partial terms, constraints and residual unconstrained ordination. There is no guarantee that these components will have the same internal names as currently, and you should be cautious when developing scripts and functions that directly access these components.

The constrained ordination algorithm is based on QR decomposition of constraints and conditions (environmental data), and the QR component is saved separately for partial and constrained components. The QR decomposition of constraints can be accessed with `qr.cca`. This will also include the residual effects of partial terms (Conditions), and it should be used together with `ordiYbar(x, "partial")`. 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`. See `qr` for other possible uses of this component. For instance, the rank of the constraints can be found from the QR decomposition.

The eigenvalues of the solution can be accessed with `eigenvals.cca`. Eigenvalues are not evaluated for partial component, and they will only be available for constrained and residual components.

The ordination scores are internally stored as (weighted) orthonormal scores matrices. These results can be accessed with `scores.cca` and `scores.rda` functions. The ordination scores are scaled when accessed with `scores` functions, but internal (weighted) orthonormal scores can be accessed by setting `scaling = FALSE`. Unconstrained residual component has species and site scores, and constrained component has also fitted site scores or linear combination scores for sites and biplot scores and centroids for constraint variables. The biplot scores correspond to the `model.matrix`, and centroids are calculated for factor variables when they were used. The scores can be selected by defining the axes, and there is no direct way of accessing all scores of a certain component. The number of dimensions can be assessed from `eigenvals`. In addition, some other types can be derived from the results although not saved in the results. For instance, regression scores and model coefficients can be accessed with `scores` and `coef` functions. Partial component will have no scores.

Distance-based methods (`dbrda`, `capscale`) can have negative eigenvalues and associated imaginary axis scores. In addition, species scores are initially missing in `dbrda` and they are accessory and found after analysis in `capscale` (and may be misleading). Function `sppscores` can be used to add species scores or replace them with more meaningful ones.

Note

The latest large change of result object was made in release 2.5-1 in 2016. You can modernize ancient stray results with `modernobject <- update(ancientobject)`.

Author(s)

Jari Oksanen

References

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English ed. Elsevier.

See Also

The core function is `ordConstrained` which is called by `cca`, `rda`, `dbrda`, `capscale` as well as by unconstrained methods `pca`, `ca` and `pco`. The basic class is `"cca"` for all methods, and the following functions are defined for this class: `add1.cca`, `alias.cca`, `anova.cca`, `as.mlm.cca`, `biplot.cca`, `bstick.cca`, `calibrate.cca`, `coef.cca`, `cooks.distance.cca`, `deviance.cca`, `df.residual.cca`, `drop1.cca`, `eigenvals.cca`, `estVar.cca`, `extractAIC.cca`, `fitted.cca`, `goodness.cca`, `hatvalues.cca`, `influence.cca`, `labels.cca`, `model.frame.cca`, `model.matrix.cca`, `nobs.cca`, `permutest.cca`, `plot.cca`, `points.cca`, `predict.cca`, `print.cca`, `qr.cca`, `residuals.cca`, `RsquareAdj.cca`, `rstandard.cca`, `rstudent.cca`, `scores.cca`, `screeplot.cca`, `sigma.cca`, `simulate.cca`, `SSD.cca`, `stressplot.cca`, `summary.cca`, `text.cca`, `tolerance.cca`, `vcov.cca`, `weights.cca`. Other functions handling `"cca"` objects include `inertcomp`, `intersetcor`, `mso`, `ordistep`, `ordiR2step` and `vif.cca`. Functions that can be regarded as special cases of `"cca"` methods include `adonis2` and `varpart`.

Examples

```
data(dune, dune.env)
mod0 <- cca(dune ~ Management + Moisture, dune.env)
## refit the model in ordConstrained
mrefit <- ordConstrained(ordiYbar(mod0, "initial"),
                        model.matrix(mod0),
                        NULL,
                        "pass")

## implement weighted RDA (weights w)
initWPCA <- function(Y, w) {
  Y <- as.matrix(Y)
  Y <- .Call("do_wcentre", Y, w)
  attr(Y, "RW") <- w
  attr(Y, "METHOD") <- "WPCA"
  Y
}
mod <- rda(dune ~ Management + Moisture, dune.env)
mod
w <- rowSums(dune)/sum(dune)
adjust <- 1 + 1/(nrow(dune) - 1)
wmod <- ordConstrained(initWPCA(dune, w * adjust),
                        model.matrix(~ Management + Moisture,dune.env)[,-1],
```

```
                        NULL, "pass")
class(wmod) <- c("wrda", "cca") # for clean print.cca
wmod

## CCA as weighted RDA (species scores differ, though).
dchi <- decostand(dune, "chi.sq")
ccmod <- ordConstrained(initWPCA(dchi, w), model.matrix(mod),
                        NULL, "pass")
class(ccmod) <- c("wrda", "cca")
ccmod # wrda
mod0  # cca
```

CCorA *Canonical Correlation Analysis*

Description

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

Usage

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

## S3 method for class 'CCorA'
biplot(x, plot.type="ov", xlabs, plot.axes = 1:2, int=0.5,
   col.Y="red", col.X="blue", cex=c(0.7,0.9), ...)
```

Arguments

`Y`	Left matrix (object class: `matrix` or `data.frame`).
`X`	Right matrix (object class: `matrix` or `data.frame`).
`stand.Y`	Logical; should `Y` be standardized?
`stand.X`	Logical; should `X` be standardized?
`permutations`	a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
`x`	`CCoaR` result object.
`plot.type`	A character string indicating which of the following plots should be produced: `"objects"`, `"variables"`, `"ov"` (separate graphs for objects and variables), or `"biplots"`. Any unambiguous subset containing the first letters of these names can be used instead of the full names.
`xlabs`	Row labels. The default is to use row names, `NULL` uses row numbers instead, and `NA` suppresses plotting row names completely.

`plot.axes`	A vector with 2 values containing the order numbers of the canonical axes to be plotted. Default: first two axes.
`int`	Radius of the inner circles plotted as visual references in the plots of the variables. Default: `int=0.5`. With `int=0`, no inner circle is plotted.
`col.Y`	Color used for objects and variables in the first data table (Y) plots. In biplots, the objects are in black.
`col.X`	Color used for objects and variables in the second data table (X) plots.
`cex`	A vector with 2 values containing the size reduction factors for the object and variable names, respectively, in the plots. Default values: `cex=c(0.7,0.9)`.
`...`	Other arguments passed to these functions. The function `biplot.CCorA` passes graphical arguments to `biplot` and `biplot.default`. `CCorA` currently ignores extra arguments.

Details

Canonical correlation analysis (Hotelling 1936) seeks linear combinations of the variables of `Y` that are maximally correlated to linear combinations of the variables of `X`. The analysis estimates the relationships and displays them in graphs. Pillai's trace statistic is computed and tested parametrically (F-test); a permutation test is also available.

Algorithmic note – The blunt approach would be to read the two matrices, compute the covariance matrices, then the matrix `S12 %*% inv(S22) %*% t(S12) %*% inv(S11)`. Its trace is Pillai's trace statistic. This approach may fail, however, when there is heavy multicollinearity in very sparse data matrices. The safe approach is to replace all data matrices by their PCA object scores.

The function can produce different types of plots depending on the option chosen: `"objects"` produces two plots of the objects, one in the space of Y, the second in the space of X; `"variables"` produces two plots of the variables, one of the variables of Y in the space of Y, the second of the variables of X in the space of X; `"ov"` produces four plots, two of the objects and two of the variables; `"biplots"` produces two biplots, one for the first matrix (Y) and one for second matrix (X) solutions. For biplots, the function passes all arguments to `biplot.default`; consult its help page for configuring biplots.

Value

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

`Pillai`	Pillai's trace statistic = sum of the canonical eigenvalues.
`Eigenvalues`	Canonical eigenvalues. They are the squares of the canonical correlations.
`CanCorr`	Canonical correlations.
`Mat.ranks`	Ranks of matrices `Y` and `X`.
`RDA.Rsquares`	Bimultivariate redundancy coefficients (R-squares) of RDAs of Y|X and X|Y.
`RDA.adj.Rsq`	`RDA.Rsquares` adjusted for `n` and the number of explanatory variables.
`nperm`	Number of permutations.
`p.Pillai`	Parametric probability value associated with Pillai's trace.
`p.perm`	Permutational probability associated with Pillai's trace.

Cy	Object scores in Y biplot.
Cx	Object scores in X biplot.
corr.Y.Cy	Scores of Y variables in Y biplot, computed as cor(Y,Cy).
corr.X.Cx	Scores of X variables in X biplot, computed as cor(X,Cx).
corr.Y.Cx	cor(Y,Cy) available for plotting variables Y in space of X manually.
corr.X.Cy	cor(X,Cx) available for plotting variables X in space of Y manually.
control	A list of control values for the permutations as returned by the function how.
call	Call to the CCorA function.

Author(s)

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

References

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

Legendre, P. 2005. Species associations: the Kendall coefficient of concordance revisited. *Journal of Agricultural, Biological, and Environmental Statistics* **10**: 226-245.

Examples

```
# Example using two mite groups. The mite data are available in vegan
data(mite)
# Two mite species associations (Legendre 2005, Fig. 4)
group.1 <- c(1,2,4:8,10:15,17,19:22,24,26:30)
group.2 <- c(3,9,16,18,23,25,31:35)
# Separate Hellinger transformations of the two groups of species
mite.hel.1 <- decostand(mite[,group.1], "hel")
mite.hel.2 <- decostand(mite[,group.2], "hel")
rownames(mite.hel.1) = paste("S",1:nrow(mite),sep="")
rownames(mite.hel.2) = paste("S",1:nrow(mite),sep="")
out <- CCorA(mite.hel.1, mite.hel.2)
out
biplot(out, "ob")                 # Two plots of objects
biplot(out, "v", cex=c(0.7,0.6))  # Two plots of variables
biplot(out, "ov", cex=c(0.7,0.6)) # Four plots (2 for objects, 2 for variables)
biplot(out, "b", cex=c(0.7,0.6))  # Two biplots
biplot(out, xlabs = NA, plot.axes = c(3,5))    # Plot axes 3, 5. No object names
biplot(out, plot.type="biplots", xlabs = NULL) # Replace object names by numbers

# Example using random numbers. No significant relationship is expected
mat1 <- matrix(rnorm(60),20,3)
mat2 <- matrix(rnorm(100),20,5)
out2 = CCorA(mat1, mat2, permutations=99)
out2
biplot(out2, "b")
```

clamtest *Multinomial Species Classification Method (CLAM)*

Description

The CLAM statistical approach for classifying generalists and specialists in two distinct habitats is described in Chazdon et al. (2011).

Usage

```
clamtest(comm, groups, coverage.limit = 10, specialization = 2/3,
   npoints = 20, alpha = 0.05/20)
## S3 method for class 'clamtest'
summary(object, ...)
## S3 method for class 'clamtest'
plot(x, xlab, ylab, main,  pch = 21:24, col.points = 1:4,
   col.lines = 2:4, lty = 1:3, position = "bottomright", ...)
```

Arguments

`comm`	Community matrix, consisting of counts.
`groups`	A vector identifying the two habitats. Must have exactly two unique values or levels. Habitat IDs in the grouping vector must match corresponding rows in the community matrix `comm`.
`coverage.limit`	Integer, the sample coverage based correction is applied to rare species with counts below this limit. Sample coverage is calculated separately for the two habitats. Sample relative abundances are used for species with higher than or equal to `coverage.limit` total counts per habitat.
`specialization`	Numeric, specialization threshold value between 0 and 1. The value of $2/3$ represents 'supermajority' rule, while a value of $1/2$ represents a 'simple majority' rule to assign shared species as habitat specialists.
`npoints`	Integer, number of points used to determine the boundary lines in the plots.
`alpha`	Numeric, nominal significance level for individual tests. The default value reduces the conventional limit of 0.05 to account for overdispersion and multiple testing for several species simultaneously. However, the is no firm reason for exactly this limit.
`x`, `object`	Fitted model object of class `"clamtest"`.
`xlab`, `ylab`	Labels for the plot axes.
`main`	Main title of the plot.
`pch`, `col.points`	Symbols and colors used in plotting species groups.
`lty`, `col.lines`	Line types and colors for boundary lines in plot to separate species groups.
`position`	Position of figure legend, see `legend` for specification details. Legend not shown if `position = NULL`.
`...`	Additional arguments passed to methods.

Details

The method uses a multinomial model based on estimated species relative abundance in two habitats (A, B). It minimizes bias due to differences in sampling intensities between two habitat types as well as bias due to insufficient sampling within each habitat. The method permits a robust statistical classification of habitat specialists and generalists, without excluding rare species *a priori* (Chazdon et al. 2011). Based on a user-defined `specialization` threshold, the model classifies species into one of four groups: (1) generalists; (2) habitat A specialists; (3) habitat B specialists; and (4) too rare to classify with confidence.

Value

A data frame (with class attribute `"clamtest"`), with columns:

`Species:`	species name (column names from `comm`),
`Total_*A*:`	total count in habitat A,
`Total_*B*:`	total count in habitat B,
`Classes:`	species classification, a factor with levels `Generalist`, `Specialist_*A*`, `Specialist_*B*`, and `Too_rare`.

`*A*` and `*B*` are placeholders for habitat names/labels found in the data.

The `summary` method returns descriptive statistics of the results. The `plot` method returns values invisibly and produces a bivariate scatterplot of species total abundances in the two habitats. Symbols and boundary lines are shown for species groups.

Note

The code was tested against standalone CLAM software provided on the website of Anne Chao (which were then at http://chao.stat.nthu.edu.tw/wordpress); minor inconsistencies were found, especially for finding the threshold for 'too rare' species. These inconsistencies are probably due to numerical differences between the two implementation. The current R implementation uses root finding for iso-lines instead of iterative search.

The original method (Chazdon et al. 2011) has two major problems:

1. It assumes that the error distribution is multinomial. This is a justified choice if individuals are freely distributed, and there is no over-dispersion or clustering of individuals. In most ecological data, the variance is much higher than multinomial assumption, and therefore test statistic are too optimistic.
2. The original authors suggest that multiple testing adjustment for multiple testing should be based on the number of points (`npoints`) used to draw the critical lines on the plot, whereas the adjustment should be based on the number of tests (i.e., tested species). The function uses the same numerical values as the original paper, but there is no automatic connection between `npoints` and `alpha` arguments, but you must work out the adjustment yourself.

Author(s)

Peter Solymos <solymos@ualberta.ca>

References

Chazdon, R. L., Chao, A., Colwell, R. K., Lin, S.-Y., Norden, N., Letcher, S. G., Clark, D. B., Finegan, B. and Arroyo J. P.(2011). A novel statistical method for classifying habitat generalists and specialists. *Ecology* **92**, 1332–1343.

Examples

```
data(mite)
data(mite.env)
sol <- with(mite.env, clamtest(mite, Shrub=="None", alpha=0.005))
summary(sol)
head(sol)
plot(sol)
```

`commsim` *Create an Object for Null Model Algorithms*

Description

The `commsim` function can be used to feed Null Model algorithms into `nullmodel` analysis. The `make.commsim` function returns various predefined algorithm types (see Details). These functions represent low level interface for community null model infrastructure in **vegan** with the intent of extensibility, and less emphasis on direct use by users.

Usage

```
commsim(method, fun, binary, isSeq, mode)
make.commsim(method)
## S3 method for class 'commsim'
print(x, ...)
```

Arguments

`method`	Character, name of the algorithm.
`fun`	A function. For possible formal arguments of this function see Details.
`binary`	Logical, if the algorithm applies to presence-absence or count matrices.
`isSeq`	Logical, if the algorithm is sequential (needs burnin and thinning) or not.
`mode`	Character, storage mode of the community matrix, either `"integer"` or `"double"`.
`x`	An object of class `commsim`.
`...`	Additional arguments.

Details

The function `fun` must return an array of `dim(nr, nc, n)`, and must take some of the following arguments:

- `x:` input matrix,
- `n:` number of permuted matrices in output,
- `nr:` number of rows,
- `nc:` number of columns,
- `rs:` vector of row sums,
- `cs:` vector of column sums,
- `rf:` vector of row frequencies (non-zero cells),
- `cf:` vector of column frequencies (non-zero cells),
- `s:` total sum of `x`,
- `fill:` matrix fill (non-zero cells),
- `thin:` thinning value for sequential algorithms,
- `...:` additional arguments.

You can define your own null model, but several null model algorithm are pre-defined and can be called by their name. The predefined algorithms are described in detail in the following chapters. The binary null models produce matrices of zeros (absences) and ones (presences) also when input matrix is quantitative. There are two types of quantitative data: Counts are integers with a natural unit so that individuals can be shuffled, but abundances can have real (floating point) values and do not have a natural subunit for shuffling. All quantitative models can handle counts (integers), but only some are able to handle real values. Some of the null models are sequential so that the next matrix is derived from the current one. This makes models dependent from previous models, and usually you must thin these matrices and study the sequences for stability: see `oecosimu` for details and instructions.

See Examples for structural constraints imposed by each algorithm and defining your own null model.

Value

An object of class `commsim` with elements corresponding to the arguments (`method`, `binary`, `isSeq`, `mode`, `fun`).

If the input of `make.comsimm` is a `commsim` object, it is returned without further evaluation. If this is not the case, the character `method` argument is matched against predefined algorithm names. An error message is issued if none such is found. If the `method` argument is missing, the function returns names of all currently available null model algorithms as a character vector.

Binary null models

All binary null models preserve fill: number of presences or conversely the number of absences. The classic models may also preserve column (species) frequencies (`c0`) or row frequencies or species richness of each site (`r0`) and take into account commonness and rarity of species (`r1`, `r2`). Algorithms `swap`, `tswap`, `curveball`, `quasiswap` and `backtrack` preserve both row and column frequencies. Three first ones are sequential but the two latter are non-sequential and produce independent matrices. Basic algorithms are reviewed by Wright et al. (1998).

`"r00"`: non-sequential algorithm for binary matrices that only preserves the number of presences (fill).

`"r0"`: non-sequential algorithm for binary matrices that preserves the site (row) frequencies.

`"r1"`: non-sequential algorithm for binary matrices that preserves the site (row) frequencies, but uses column marginal frequencies as probabilities of selecting species.

`"r2"`: non-sequential algorithm for binary matrices that preserves the site (row) frequencies, and uses squared column marginal frequencies as as probabilities of selecting species.

`"c0"`: non-sequential algorithm for binary matrices that preserves species frequencies (Jonsson 2001).

`"swap"`: sequential algorithm for binary matrices that changes the matrix structure, but does not influence marginal sums (Gotelli & Entsminger 2003). This inspects 2×2 submatrices so long that a swap can be done. Some arbitrary matrix with (nearly) complete fill cannot be swapped; function `nestedchecker` gives the number of swappable submatrices. This also applies to other `swap` models.

`"tswap"`: sequential algorithm for binary matrices. Same as the `"swap"` algorithm, but it tries a fixed number of times and performs zero to many swaps at one step (according to the thin argument in the call). This approach was suggested by Miklós & Podani (2004) because they found that ordinary swap may lead to biased sequences, since some columns or rows are more easily swapped.

`"curveball"`: sequential method for binary matrices that implements the 'Curveball' algorithm of Strona et al. (2014). The algorithm selects two random rows and finds the set of unique species that occur only in one of these rows. The algorithm distributes the set of unique species to rows preserving the original row frequencies. Zero to several species are swapped in one step, and usually the matrix is perturbed more strongly than in other sequential methods.

`"quasiswap"`: non-sequential algorithm for binary matrices that implements a method where matrix is first filled honouring row and column totals, but with integers that may be larger than one. Then the method inspects random 2×2 matrices and performs a quasiswap on them. In addition to ordinary swaps, quasiswap can reduce numbers above one to ones preserving marginal totals (Miklós & Podani 2004). The method is non-sequential, but it accepts `thin` argument: the convergence is checked at every `thin` steps. This allows performing several ordinary swaps in addition to fill changing swaps which helps in reducing or removing the bias.

`"greedyqswap"`: A greedy variant of quasiswap. In greedy step, one element of the 2×2 matrix is taken from > 1 elements. The greedy steps are biased, but the method can be thinned, and only the first of `thin` steps is greedy. Even modest thinning (say `thin = 20`) removes or reduces the bias, and `thin = 100` (1% greedy steps) looks completely safe and still speeds up simulation. The code is experimental and it is provided here for further scrutiny, and should be tested for bias before use.

`"backtrack"`: non-sequential algorithm for binary matrices that implements a filling method with constraints both for row and column frequencies (Gotelli & Entsminger 2001). The matrix is first filled randomly, but typically row and column sums are reached before all incidences are filled in. After this the function "backtracks" removing some incidences and starting to fill again. This backtracking is done so many times that all incidences will be filled into matrix. The results may be biased and should be inspected carefully before use.

Quantitative Models for Counts with Fixed Marginal Sums

These models shuffle individuals of counts and keep marginal sums fixed, but marginal frequencies are not preserved. Algorithm `r2dtable` uses standard R function `r2dtable` also used for simulated P-values in `chisq.test`. Algorithm `quasiswap_count` uses the same, but preserves the original fill. Typically this means increasing numbers of zero cells and the result is zero-inflated with respect to `r2dtable`.

`"r2dtable"`: non-sequential algorithm for count matrices. This algorithm keeps matrix sum and row/column sums constant. Based on `r2dtable`.

`"quasiswap_count"`: non-sequential algorithm for count matrices. This algorithm is similar as Carsten Dormann's `swap.web` function in the package **bipartite**. First, a random matrix is generated by the `r2dtable` function preserving row and column sums. Then the original matrix fill is reconstructed by sequential steps to increase or decrease matrix fill in the random matrix. These steps are based on swapping 2×2 submatrices (see `"swap_count"` algorithm for details) to maintain row and column totals.

Quantitative Swap Models

Quantitative swap models are similar to binary `swap`, but they swap the largest permissible value. The models in this section all maintain the fill and perform a quantitative swap only if this can be done without changing the fill. Single step of swap often changes the matrix very little. In particular, if cell counts are variable, high values change very slowly. Checking the chain stability and independence is even more crucial than in binary swap, and very strong `thinning` is often needed. These models should never be used without inspecting their properties for the current data. These null models can also be defined using `permatswap` function.

`"swap_count"`: sequential algorithm for count matrices. This algorithm find 2×2 submatrices that can be swapped leaving column and row totals and fill unchanged. The algorithm finds the largest value in the submatrix that can be swapped (d). Swap means that the values in diagonal or antidiagonal positions are decreased by d, while remaining cells are increased by d. A swap is made only if fill does not change.

`"abuswap_r"`: sequential algorithm for count or nonnegative real valued matrices with fixed row frequencies (see also `permatswap`). The algorithm is similar to `swap_count`, but uses different swap value for each row of the 2×2 submatrix. Each step changes the the corresponding column sums, but honours matrix fill, row sums, and row/column frequencies (Hardy 2008; randomization scheme 2x).

`"abuswap_c"`: sequential algorithm for count or nonnegative real valued matrices with fixed column frequencies (see also `permatswap`). The algorithm is similar as the previous one, but operates on columns. Each step changes the the corresponding row sums, but honours matrix fill, column sums, and row/column frequencies (Hardy 2008; randomization scheme 3x).

Quantitative Swap and Shuffle Models

Quantitative Swap and Shuffle methods (`swsh` methods) preserve fill and column and row frequencies, and also either row or column sums. The methods first perform a binary `quasiswap` and then shuffle original quantitative data to non-zero cells. The `samp` methods shuffle original non-zero cell values and can be used also with non-integer data. The `both` methods redistribute individuals randomly among non-zero cells and can only be used with integer data. The shuffling is either free

over the whole matrix, or within rows (r methods) or within columns (c methods). Shuffling within a row preserves row sums, and shuffling within a column preserves column sums. These models can also be defined with permatswap.

"swsh_samp": non-sequential algorithm for quantitative data (either integer counts or non-integer values). Original non-zero values values are shuffled.

"swsh_both": non-sequential algorithm for count data. Individuals are shuffled freely over non-zero cells.

"swsh_samp_r": non-sequential algorithm for quantitative data. Non-zero values (samples) are shuffled separately for each row.

"swsh_samp_c": non-sequential algorithm for quantitative data. Non-zero values (samples) are shuffled separately for each column.

"swsh_both_r": non-sequential algorithm for count matrices. Individuals are shuffled freely for non-zero values within each row.

"swsh_both_c": non-sequential algorithm for count matrices. Individuals are shuffled freely for non-zero values with each column.

Quantitative Shuffle Methods

Quantitative shuffle methods are generalizations of binary models r00, r0 and c0. The _ind methods shuffle individuals so that the grand sum, row sum or column sums are preserved. These methods are similar as r2dtable but with still slacker constraints on marginal sums. The _samp and _both methods first apply the corresponding binary model with similar restriction on marginal frequencies and then distribute quantitative values over non-zero cells. The _samp models shuffle original cell values and can therefore handle also non-count real values. The _both models shuffle individuals among non-zero values. The shuffling is over the whole matrix in r00_, and within row in r0_ and within column in c0_ in all cases.

"r00_ind": non-sequential algorithm for count matrices. This algorithm preserves grand sum and individuals are shuffled among cells of the matrix.

"r0_ind": non-sequential algorithm for count matrices. This algorithm preserves row sums and individuals are shuffled among cells of each row of the matrix.

"c0_ind": non-sequential algorithm for count matrices. This algorithm preserves column sums and individuals are shuffled among cells of each column of the matrix.

"r00_samp": non-sequential algorithm for count or nonnegative real valued (mode = "double") matrices. This algorithm preserves grand sum and cells of the matrix are shuffled.

"r0_samp": non-sequential algorithm for count or nonnegative real valued (mode = "double") matrices. This algorithm preserves row sums and cells within each row are shuffled.

"c0_samp": non-sequential algorithm for count or nonnegative real valued (mode = "double") matrices. This algorithm preserves column sums constant and cells within each column are shuffled.

"r00_both": non-sequential algorithm for count matrices. This algorithm preserves grand sum and cells and individuals among cells of the matrix are shuffled.

"r0_both": non-sequential algorithm for count matrices. This algorithm preserves grand sum and cells and individuals among cells of each row are shuffled.

"c0_both": non-sequential algorithm for count matrices. This algorithm preserves grand sum and cells and individuals among cells of each column are shuffled.

Author(s)

Jari Oksanen and Peter Solymos

References

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

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

Hardy, O. J. (2008) Testing the spatial phylogenetic structure of local communities: statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology* 96, 914–926.

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

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

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

Strona, G., Nappo, D., Boccacci, F., Fattorini, S. & San-Miguel-Ayanz, J. (2014). A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. *Nature Communications* 5:4114 doi:10.1038/ncomms5114.

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

See Also

See `permatfull`, `permatswap` for alternative specification of quantitative null models. Function `oecosimu` gives a higher-level interface for applying null models in hypothesis testing and analysis of models. Function `nullmodel` and `simulate.nullmodel` are used to generate arrays of simulated null model matrices.

Examples

```
## write the r00 algorithm
f <- function(x, n, ...)
    array(replicate(n, sample(x)), c(dim(x), n))
(cs <- commsim("r00", fun=f, binary=TRUE,
    isSeq=FALSE, mode="integer"))

## retrieving the sequential swap algorithm
(cs <- make.commsim("swap"))

## feeding a commsim object as argument
make.commsim(cs)

## making the missing c1 model using r1 as a template
##   non-sequential algorithm for binary matrices
##   that preserves the species (column) frequencies,
```

```
##   but uses row marginal frequencies
##   as probabilities of selecting sites
f <- function (x, n, nr, nc, rs, cs, ...) {
    out <- array(0L, c(nr, nc, n))
    J <- seq_len(nc)
    storage.mode(rs) <- "double"
    for (k in seq_len(n))
        for (j in J)
            out[sample.int(nr, cs[j], prob = rs), j, k] <- 1L
    out
}
cs <- make.commsim("r1")
cs$method <- "c1"
cs$fun <- f

## structural constraints
diagfun <- function(x, y) {
    c(sum = sum(y) == sum(x),
        fill = sum(y > 0) == sum(x > 0),
        rowSums = all(rowSums(y) == rowSums(x)),
        colSums = all(colSums(y) == colSums(x)),
        rowFreq = all(rowSums(y > 0) == rowSums(x > 0)),
        colFreq = all(colSums(y > 0) == colSums(x > 0)))
}
evalfun <- function(meth, x, n) {
    m <- nullmodel(x, meth)
    y <- simulate(m, nsim=n)
    out <- rowMeans(sapply(1:dim(y)[3],
        function(i) diagfun(attr(y, "data"), y[,,i])))
    z <- as.numeric(c(attr(y, "binary"), attr(y, "isSeq"),
        attr(y, "mode") == "double"))
    names(z) <- c("binary", "isSeq", "double")
    c(z, out)
}
x <- matrix(rbinom(10*12, 1, 0.5)*rpois(10*12, 3), 12, 10)
algos <- make.commsim()
a <- t(sapply(algos, evalfun, x=x, n=10))
print(as.table(ifelse(a==1,1,0)), zero.print = ".")
```

`contribdiv` *Contribution Diversity Approach*

Description

The contribution diversity approach is based in the differentiation of within-unit and among-unit diversity by using additive diversity partitioning and unit distinctiveness.

Usage

```
contribdiv(comm, index = c("richness", "simpson"),
```

```
     relative = FALSE, scaled = TRUE, drop.zero = FALSE)
## S3 method for class 'contribdiv'
plot(x, sub, xlab, ylab, ylim, col, ...)
```

Arguments

comm	The community data matrix with samples as rows and species as column.
index	Character, the diversity index to be calculated.
relative	Logical, if TRUE then contribution diversity values are expressed as their signed deviation from their mean. See details.
scaled	Logical, if TRUE then relative contribution diversity values are scaled by the sum of gamma values (if index = "richness") or by sum of gamma values times the number of rows in comm (if index = "simpson"). See details.
drop.zero	Logical, should empty rows dropped from the result? If empty rows are not dropped, their corresponding results will be NAs.
x	An object of class "contribdiv".
sub, xlab, ylab, ylim, col	Graphical arguments passed to plot.
...	Other arguments passed to plot.

Details

This approach was proposed by Lu et al. (2007). Additive diversity partitioning (see adipart for more references) deals with the relation of mean alpha and the total (gamma) diversity. Although alpha diversity values often vary considerably. Thus, contributions of the sites to the total diversity are uneven. This site specific contribution is measured by contribution diversity components. A unit that has e.g. many unique species will contribute more to the higher level (gamma) diversity than another unit with the same number of species, but all of which common.

Distinctiveness of species j can be defined as the number of sites where it occurs (n_j), or the sum of its relative frequencies (p_j). Relative frequencies are computed sitewise and $sum_j p_i j$s at site i sum up to 1.

The contribution of site i to the total diversity is given by $alpha_i = sum_j(1/n_i j)$ when dealing with richness and $alpha_i = sum(p_{ij} * (1 - p_{ij}))$ for the Simpson index.

The unit distinctiveness of site i is the average of the species distinctiveness, averaging only those species which occur at site i. For species richness: $alpha_i = mean(n_i)$ (in the paper, the second equation contains a typo, n is without index). For the Simpson index: $alpha_i = mean(n_i)$.

The Lu et al. (2007) gives an in-depth description of the different indices.

Value

An object of class "contribdiv" inheriting from data frame.

Returned values are alpha, beta and gamma components for each sites (rows) of the community matrix. The "diff.coef" attribute gives the differentiation coefficient (see Examples).

Author(s)

Péter Sólymos, <solymos@ualberta.ca>

References

Lu, H. P., Wagner, H. H. and Chen, X. Y. 2007. A contribution diversity approach to evaluate species diversity. *Basic and Applied Ecology*, 8, 1–12.

See Also

adipart, diversity

Examples

```
## Artificial example given in
## Table 2 in Lu et al. 2007
x <- matrix(c(
1/3,1/3,1/3,0,0,0,
0,0,1/3,1/3,1/3,0,
0,0,0,1/3,1/3,1/3),
3, 6, byrow = TRUE,
dimnames = list(LETTERS[1:3],letters[1:6]))
x
## Compare results with Table 2
contribdiv(x, "richness")
contribdiv(x, "simpson")
## Relative contribution (C values), compare with Table 2
(cd1 <- contribdiv(x, "richness", relative = TRUE, scaled = FALSE))
(cd2 <- contribdiv(x, "simpson", relative = TRUE, scaled = FALSE))
## Differentiation coefficients
attr(cd1, "diff.coef") # D_ST
attr(cd2, "diff.coef") # D_DT
## BCI data set
data(BCI)
opar <- par(mfrow=c(2,2))
plot(contribdiv(BCI, "richness"), main = "Absolute")
plot(contribdiv(BCI, "richness", relative = TRUE), main = "Relative")
plot(contribdiv(BCI, "simpson"))
plot(contribdiv(BCI, "simpson", relative = TRUE))
par(opar)
```

---

dbrda *Principal Coordinates Analysis and [Partial] Distance-based Redundancy Analysis*

Description

Distance-based redundancy analysis (dbRDA) 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 dbRDA will be less efficient. Functions dbrda is constrained versions of metric scaling, a.k.a. principal coordinates analysis, which are based on the Euclidean distance but can be used, and are more useful, with other dissimilarity

measures. Function capscale is a simplified version based on Euclidean approximation of dissimilarities. The functions can also perform unconstrained principal coordinates analysis (PCO), optionally using extended dissimilarities. pco() is a wrapper to dbrda(), which performs PCO.

Usage

```
dbrda(formula, data, distance = "euclidean", sqrt.dist = FALSE,
    add = FALSE, dfun = vegdist, metaMDSdist = FALSE,
    na.action = na.fail, subset = NULL, ...)
capscale(formula, data, distance = "euclidean", sqrt.dist = FALSE,
    comm = NULL, add = FALSE,  dfun = vegdist, metaMDSdist = FALSE,
    na.action = na.fail, subset = NULL, ...)
pco(X, ...)
```

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 or function given in dfun will be used to find the dissimilarities. The RHS defines the constraints. The constraints can be continuous variables or factors, they can be transformed within the formula, and they can have interactions as in a typical formula. The RHS can have a special term Condition that defines variables to be "partialled out" before constraints, just like in rda or cca. This allows the use of partial dbRDA.
X	Community data matrix.
data	Data frame containing the variables on the right hand side of the model formula.
distance	The name of the dissimilarity (or distance) index if the LHS of the formula is a data frame instead of dissimilarity matrix.
sqrt.dist	Take square roots of dissimilarities. See section Details below.
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 unavailable when dissimilarities were supplied. N.B., this is only available in capscale: dbrda does not return species scores. Function sppscores can be used to add species scores if they are missing.
add	Add a constant to the non-diagonal dissimilarities such that all eigenvalues are non-negative in the underlying Principal Co-ordinates Analysis (see wcmdscale for details). "lingoes" (or TRUE) uses the recommended method of Legendre & Anderson (1999: "method 1") and "cailliez" uses their "method 2". The latter is the only one in cmdscale.
dfun	Distance or dissimilarity function used. Any function returning standard "dist" and taking the index name as the first argument can be used.
metaMDSdist	Use metaMDSdist similarly as in metaMDS. This means automatic data transformation and using extended flexible shortest path dissimilarities (function stepacross) when there are many dissimilarities based on no shared species.

`na.action` Handling of missing values in constraints or conditions. The default (`na.fail`) is to stop with missing values. Choices `na.omit` and `na.exclude` delete rows with missing values, but differ in representation of results. With `na.omit` only non-missing site scores are shown, but `na.exclude` gives `NA` for scores of missing observations. Unlike in `rda`, no WA scores are available for missing constraints or conditions.

`subset` Subset of data rows. This can be a logical vector which is `TRUE` for kept observations, or a logical expression which can contain variables in the working environment, `data` or species names of the community data (if given in the formula or as `comm` argument).

`...` Other parameters passed to underlying functions (e.g., `metaMDSdist`). For `pco()` argument are passed to `dbrda()`.

Details

Functions `dbrda` and `capscale` provide two alternative implementations of dbRDA. Function `dbrda` is based on McArdle & Anderson (2001) and directly decomposes dissimilarities. With Euclidean distances results are identical to `rda`. Non-Euclidean dissimilarities may give negative eigenvalues associated with imaginary axes. Function `capscale` is based on Legendre & Anderson (1999): the dissimilarity data are first ordinated using metric scaling, and the ordination results are analysed as `rda`. `capscale` ignores the imaginary component and will not give negative eigenvalues (but will report the magnitude on imaginary component).

If the user supplied a community data frame instead of dissimilarities, the functions will find dissimilarities using `vegdist` or distance function given in `dfun` with specified `distance`. The functions will accept distance objects from `vegdist`, `dist`, or any other method producing compatible objects. 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" analysis can be performed.

Function `dbrda` does not return species scores, and they can also be missing in `capscale`, but they can be added after the analysis using function `sppscores`.

Non-Euclidean dissimilarities can produce negative eigenvalues (Legendre & Anderson 1999, McArdle & Anderson 2001). If there are negative eigenvalues, the printed output of `capscale` will add a column with sums of positive eigenvalues and an item of sum of negative eigenvalues, and `dbrda` will add a column giving the number of real dimensions with positive eigenvalues. If negative eigenvalues are disturbing, functions let you distort the dissimilarities so that only non-negative eigenvalues will be produced with argument `add = TRUE`. Alternatively, with `sqrt.dist = TRUE`, square roots of dissimilarities can be used which may help in avoiding negative eigenvalues (Legendre & Anderson 1999).

The functions can be also used to perform ordinary metric scaling a.k.a. principal coordinates analysis by using a formula with only a constant on the right hand side, or `comm ~ 1`. The new function `pco()` implements principal coordinates analysis via `dbrda()` directly, using this formula. With `metaMDSdist = TRUE`, the function can do automatic data standardization and use extended dissimilarities using function `stepacross` similarly as in non-metric multidimensional scaling with `metaMDS`.

Value

The functions return an object of class dbrda or capscale which inherit from rda. See cca.object for description of the result object. Function pco() returns an object of class "vegan_pco" (which inherits from class "dbrda") to avoid clashes with other packages.

Note

Function dbrda implements real distance-based RDA and is preferred over capscale. capscale was originally developed as a variant of constrained analysis of proximities (Anderson & Willis 2003), but these developments made it more similar to dbRDA. However, it discards the imaginary dimensions with negative eigenvalues and ordination and significance tests area only based on real dimensions and positive eigenvalues. capscale may be removed from **vegan** in the future. It has been in vegan since 2003 (CRAN release 1.6-0) while dbrda was first released in 2016 (version 2.4-0), and removal of capscale may be disruptive to historical examples and scripts, but in modern times dbrda should be used.

The inertia is named after the dissimilarity index as defined in the dissimilarity data, or as unknown distance if such information is missing. If the largest original dissimilarity was larger than 4, capscale handles input similarly as rda and bases its analysis on variance instead of sum of squares. Keyword mean is added to the inertia in these cases, e.g. with Euclidean and Manhattan distances. Inertia is based on squared index, and keyword squared is added to the name of distance, unless data were square root transformed (argument sqrt.dist=TRUE). If an additive constant was used with argument add, Lingoes or Cailliez adjusted is added to the the name of inertia, and the value of the constant is printed.

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.

Gower, J.C. (1985). Properties of Euclidean and non-Euclidean distance matrices. *Linear Algebra and its Applications* 67, 81–97.

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

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

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

See Also

rda, cca, plot.cca, anova.cca, vegdist, dist, cmdscale, wcmdscale for underlying and related functions. Function sppscores can add species scores or replace existing species scores.

The function returns similar result object as rda (see cca.object). This section for rda gives a more complete list of functions that can be used to access and analyse dbRDA results.

Examples

```
data(varespec, varechem)
## dbrda
dbrda(varespec ~ N + P + K + Condition(Al), varechem, dist="bray")
## avoid negative eigenvalues with sqrt distances
dbrda(varespec ~ N + P + K + Condition(Al), varechem, dist="bray",
     sqrt.dist = TRUE)
## avoid negative eigenvalues also with Jaccard distances
(m <- dbrda(varespec ~ N + P + K + Condition(Al), varechem, dist="jaccard"))
## add species scores
sppscores(m) <- wisconsin(varespec)
## pco
pco(varespec, dist = "bray", sqrt.dist = TRUE)
```

---

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 = 0.7, 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'
scores(x, display="sites", choices=1:4,
       origin=TRUE, tidy=FALSE, ...)

downweight(veg, fraction = 5)
```

Arguments

`veg`	Community data, a matrix-like object.
`iweigh`	Downweighting of rare species (0: no).
`iresc`	Number of rescaling cycles (0: no rescaling).
`ira`	Type of analysis (0: detrended, 1: basic reciprocal averaging).
`mk`	Number of segments in rescaling.
`short`	Shortest gradient to be rescaled.
`before`	Hill's piecewise transformation: values before transformation.
`after`	Hill's piecewise transformation: values after transformation – these must correspond to values in `before`.
`x`	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. If `select` is used, labels are given only to selected items in the order they occur in the scores.
`select`	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices or names (labels) of scores.
`xlim`, `ylim`	the x and y limits (min,max) of the plot.
`fraction`	Abundance fraction where downweighting begins.
`tidy`	Return scores that are compatible with **ggvegan**: all scores are in a single `data.frame`, score type is identified by factor variable `score` (`"sites"`, `"species"`), the names by variable `label`. These scores are incompatible with conventional `plot` functions, but they can be used in **ggvegan**.
`...`	Other arguments for `plot` function.

Details

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

The curvature is removed by replacing the orthogonalization of axes with detrending. In orthogonalization successive axes are made non-correlated, but detrending should remove all systematic dependence between axes. Detrending is performed using a smoothing window on `mk` segments. The packing of sites at the ends of the gradient is undone by rescaling the axes after extraction. After rescaling, the axis is supposed to be scaled by 'SD' units, so that the average width of Gaussian

species responses is supposed to be one over whole axis. Other innovations were the piecewise linear transformation of species abundances and downweighting of rare species which were regarded to have an unduly high influence on ordination axes.

It seems that detrending actually works by twisting the ordination space, so that the results look non-curved in two-dimensional projections ('lolly paper effect'). As a result, the points usually have an easily recognized triangular or diamond shaped pattern, obviously an artefact of detrending. 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 standard deviation of species scores on axis segments (parameter `mk` has no effect, since `decorana` finds the segments internally). Function `tolerance` returns this internal criterion and can be used to assess the success of rescaling.

The `plot` method plots species and site scores. Classical `decorana` scaled the axes so that smallest site score was 0 (and smallest species score was negative), but `summary`, `plot` and `scores` use the true origin, unless `origin = FALSE`.

In addition to proper eigenvalues, the function reports 'decorana values' in detrended analysis. These 'decorana values' are the values that the legacy code of `decorana` returns as eigenvalues. They are estimated during iteration, and describe the joint effects of axes and detrending. The 'decorana values' are estimated before rescaling and do not show its effect on eigenvalues. The proper eigenvalues are estimated after extraction of the axes and they are the ratio of weighted sum of squares of site and species scores even in detrended and rescaled solutions. These eigenvalues are estimated for each axis separately, but they are not additive, because higher `decorana` axes can show effects already explained by prior axes. 'Additive eigenvalues' are cleansed from the effects of prior axes, and they can be assumed to add up to total inertia (scaled Chi-square). For proportions and cumulative proportions explained you can use `eigenvals.decorana`.

Value

`decorana` returns an object of class `"decorana"`, which has `print`, `summary`, `scores`, `plot`, `points` and `text` methods, and support functions `eigenvals`, `bstick`, `screeplot`, `predict` and `tolerance`. `downweight` is an independent function that can also be used with other methods than `decorana`.

Note

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

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

Github package **natto** has an R implementation of `decorana` which allows easier inspection of the algorithm and also easier development of the function.

vegan 2.6-6 and earlier had a `summary` method, but it did nothing useful and is now defunct. All its former information can be extracted with `scores` or `weights.decorana`.

Author(s)

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

References

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

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

See Also

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

Examples

```
data(varespec)
vare.dca <- decorana(varespec)
vare.dca
plot(vare.dca, optimize = TRUE)

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

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

decostand *Standardization Methods for Community Ecology*

Description

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

Usage

```
decostand(x, method, MARGIN, range.global, logbase = 2, na.rm = FALSE, ...)
wisconsin(x, na.rm = FALSE)
decobackstand(x, zap = TRUE)
```

Arguments

x	Community data, a matrix-like object. For decobackstand standardized data.
method	Standardization method. See Details for available options.
MARGIN	Margin, if default is not acceptable. 1 = rows, and 2 = columns of x.
range.global	Matrix from which the range is found in method = "range". This allows using same ranges across subsets of data. The dimensions of MARGIN must match with x.
logbase	The logarithm base used in method = "log".
na.rm	Ignore missing values in row or column standardizations. The NA values remain as NA, but they are ignored in standardization of other values.
zap	Make near-zero values exact zeros to avoid negative values and exaggerated estimates of species richness.
...	Other arguments to the function. For methods "clr", "rclr" and "alr" see Details.

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).
- frequency: divide by margin total and multiply by the number of non-zero items, so that the average of non-zero entries is one (Oksanen 1983; default MARGIN = 2).
- normalize: make margin sum of squares equal to one. With Euclidean distances these give Chord distances (see vegdist). (default MARGIN = 1).
- range: standardize values into range 0 ... 1 (default MARGIN = 2). If all values are constant, they will be transformed to 0.

- `rank`, `rrank`: `rank` replaces abundance values by their increasing ranks leaving zeros unchanged, and `rrank` is similar but uses relative ranks with maximum 1 (default `MARGIN = 1`). Average ranks are used for tied values.
- `standardize`: scale `x` to zero mean and unit variance (default `MARGIN = 2`).
- `pa`: scale `x` to presence/absence scale (0/1).
- `chi.square`: divide by row sums and square root of column sums, and adjust for square root of matrix total (Legendre & Gallagher 2001). When used with the Euclidean distance in weighted metric scaling (principal coordinates analysis) they are Chi-square distances used in correspondence analysis; see `wcmdscale`. (default `MARGIN = 1`).
- `hellinger`: square root of `method = "total"` (Legendre & Gallagher 2001). With Euclidean distances these give Hellinger distances (see `vegdist`). (default `MARGIN = 1`).
- `log`: logarithmic transformation as suggested by Anderson et al. (2006): $\log_b(x) + 1$ for $x > 0$, where b is the base of the logarithm; zeros are left as zeros. Higher bases give less weight to quantities and more to presences, and `logbase = Inf` gives the presence/absence scaling. Please note this is *not* $\log(x + 1)$. Anderson et al. (2006) suggested this for their (strongly) modified Gower distance (implemented as `method = "altGower"` in `vegdist`), but the standardization can be used independently of distance indices.
- `alr`: Additive log ratio ("alr") transformation (Aitchison 1986) reduces data skewness and compositionality bias. The transformation assumes positive values, pseudocounts can be added with the argument `pseudocount`. One of the rows/columns is a reference that can be given by `reference` (name of index). The first row/column is used by default (`reference = 1`). Note that this transformation drops one row or column from the transformed output data. The `alr` transformation is defined formally as follows:

  $$alr = [log\frac{x_1}{x_D}, ..., log\frac{x_{D-1}}{x_D}]$$

  where the denominator sample x_D can be chosen arbitrarily. This transformation is often used with pH and other chemistry measurements. It is also commonly used as multinomial logistic regression. Default `MARGIN = 1` uses row as the `reference`.
- `clr`: centered log ratio ("clr") transformation proposed by Aitchison (1986) and it is used to reduce data skewness and compositionality bias. This transformation has frequent applications in microbial ecology (see e.g. Gloor et al., 2017). The clr transformation is defined as:

  $$clr = log\frac{x}{g(x)} = logx - logg(x)$$

  where x is a single value, and g(x) is the geometric mean of x. The method can operate only with positive data; a common way to deal with zeroes is to add pseudocount (e.g. the smallest positive value in the data), either by adding it manually to the input data, or by using the argument `pseudocount` as in `decostand(x, method = "clr", na.rm = TRUE, pseudocount = 1)`. Adding pseudocount will inevitably introduce some bias; see the `rclr` method for an alternative.
- `rclr`: robust clr ("rclr") is similar to regular clr (see above) but it allows data with zeroes. This method can avoid the use of pseudocounts, unlike the standard clr. The robust clr (rclr) the logarithmizes the data and divides it by the geometric mean of the observed features within each sample. In high dimensional data the geometric mean of rclr approximates the true

geometric mean; see e.g. Martino et al. (2019). The `rclr` transformation is defined formally as follows:

$$rclr = log \frac{x}{g(x > 0)}$$

where x is a single value, and $g(x > 0)$ is the geometric mean of sample-wide values x that are positive (> 0). The optspace algorithm performs matrix completion for the missing values that result from log transformation of the zero entries in the original input data. See optspace for more details. The following parameters can be passed to optspace through decostand: "ropt" NA to guess the rank, or a positive integer as a pre-defined rank (default: 3); "niter" maximum number of iterations allowed (default: 5); "tol" stopping criterion for reconstruction in Frobenius norm (default: 1e-5); "verbose" a logical value; TRUE to show progress, FALSE otherwise (default: FALSE); "impute" to switch on/off the matrix completion (default: impute=TRUE).

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).

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

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

Most methods will retain original absences as zeros. Exceptions are `"standardize"` where zeros of standardized values correspond to `MARGIN` average, and original zeros are negative values. The Aitchison methods `"clr"`, `"rclr"` and `"alr"` replace zeros or use `NA`. With `"clr"` the zeros are always replaced with negative values which is constant for row, but with others negative and positive values can occur both for original zeros and positive values.

Most methods will retain the rank order of input data by `MARGIN` (but not the other margin). Exceptions are `"chi.square"` and function `wisconsin` which standardize by both margins, and methods `"rclr"` and `"alr"` (and naturally `"pa"` which only has values 0 and 1).

Function `decobackstand` can be used to transform standardized data back to original. This is not possible for all standardization methods and may not be implemented to all cases where it would be possible. There are round-off errors and back-transformation is not exact, and it is wise not to overwrite the original data. With `zap=TRUE` original zeros should be exact.

Value

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

Note

Common transformations can be made with standard R functions.

Author(s)

Jari Oksanen, Etienne Laliberté (`method = "log"`), Leo Lahti (`alr`, `"clr"` and `"rclr"`).

References

Aitchison, J. The Statistical Analysis of Compositional Data (1986). London, UK: Chapman & Hall.

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

Egozcue, J.J., Pawlowsky-Glahn, V., Mateu-Figueras, G., Barcel'o-Vidal, C. (2003) Isometric logratio transformations for compositional data analysis. *Mathematical Geology* **35**, 279–300.

Gloor, G.B., Macklaim, J.M., Pawlowsky-Glahn, V. & Egozcue, J.J. (2017) Microbiome Datasets Are Compositional: And This Is Not Optional. *Frontiers in Microbiology* **8**, 2224.

Keshavan, R. H., Montanari, A., Oh, S. (2010). Matrix Completion From a Few Entries. *IEEE Transactions on Information Theory* **56**, 2980–2998.

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

Martino, C., Morton, J.T., Marotz, C.A., Thompson, L.R., Tripathi, A., Knight, R. & Zengler, K. (2019) A novel sparse compositional technique reveals microbial perturbations. *mSystems* **4**, 1.

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)

# CLR transformation for rows, with pseudocount
varespec.clr <- decostand(varespec, "clr", pseudocount = 1)

# Robust CLR (rclr) transformation for rows, no pseudocount necessary
varespec.rclr <- decostand(varespec, "rclr", impute = TRUE)

# ALR transformation for rows, with pseudocount and reference sample
varespec.alr <- decostand(varespec, "alr", pseudocount = 1, reference = 1)

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

`designdist` *Design your own Dissimilarities*

Description

Function designdist lets you define your own dissimilarities using terms for shared and total quantities, number of rows and number of columns. The shared and total quantities can be binary, quadratic or minimum terms. In binary terms, the shared component is number of shared species, and totals are numbers of species on sites. The quadratic terms are cross-products and sums of squares, and minimum terms are sums of parallel minima and row totals. Function designdist2 is similar, but finds dissimilarities among two data sets. Function chaodist lets you define your own dissimilarities using terms that are supposed to take into account the "unseen species" (see Chao et al., 2005 and Details in vegdist).

Usage

```
designdist(x, method = "(A+B-2*J)/(A+B)",
           terms = c("binary", "quadratic", "minimum"),
           abcd = FALSE, alphagamma = FALSE, name, maxdist)
designdist2(x, y, method = "(A+B-2*J)/(A+B)",
           terms = c("binary", "quadratic", "minimum"),
           abcd = FALSE, alphagamma = FALSE, name, maxdist)
chaodist(x, method = "1 - 2*U*V/(U+V)", name)
```

Arguments

x	Input data.
y	Another input data set: dissimilarities will be calculated among rows of x and rows of y.
method	Equation for your dissimilarities. This can use terms J for shared quantity, A and B for totals, N for the number of rows (sites) and P for the number of columns (species) or in chaodist it can use terms U and V. The equation can also contain any R functions that accepts vector arguments and returns vectors of the same length. It can also include functions of input x that return a scalar or a dist vector.
terms	How shared and total components are found. For vectors x and y the "quadratic" terms are J = sum(x*y), A = sum(x^2), B = sum(y^2), and "minimum" terms are J = sum(pmin(x,y)), A = sum(x) and B = sum(y), and "binary" terms are either of these after transforming data into binary form (shared number of species, and number of species for each row).
abcd	Use 2x2 contingency table notation for binary data: a is the number of shared species, b and c are the numbers of species occurring only one of the sites but not in both, and d is the number of species that occur on neither of the sites.
alphagamma	Use beta diversity notation with terms alpha for average alpha diversity for compared sites, gamma for diversity in pooled sites, and delta for the absolute value of difference of average alpha and alpha diversities of compared sites. Terms A and B refer to alpha diversities of compared sites.
name	The name you want to use for your index. The default is to combine the method equation and terms argument.

`maxdist` Theoretical maximum of the dissimilarity, or `NA` if index is open and has no absolute maximum. This is not a necessary argument, but only used in some **vegan** functions, and if you are not certain about the maximum, it is better not supply any value.

Details

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

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

The function `designdist` can implement most dissimilarity indices in `vegdist` or elsewhere, and it can also be used to implement many other indices, amongst them, most of those described in Legendre & Legendre (2012). It can also be used to implement all indices of beta diversity described in Koleff et al. (2003), but there also is a specific function `betadiver` for the purpose.

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

With `alphagamma = TRUE` it is possible to use beta diversity notation with terms `alpha` for average alpha diversity and `gamma` for gamma diversity in two compared sites. The terms are calculated as `alpha = (A+B)/2`, `gamma = A+B-J` and `delta = abs(A-B)/2`. Terms `A` and `B` are also available and give the alpha diversities of the individual compared sites. The beta diversity terms may make sense only for binary terms (so that diversities are expressed in numbers of species), but they are calculated for quadratic and minimum terms as well (with a warning).

Function `chaodist` is similar to `designgist`, but uses terms `U` and `V` of Chao et al. (2005). These terms are supposed to take into account the effects of unseen species. Both `U` and `V` are scaled to range $0 \ldots 1$. They take the place of `A` and `B` and the product `U*V` is used in the place of `J` of `designdist`. Function `chaodist` can implement any commonly used Chao et al. (2005) style dissimilarity:

`1 - 2*U*V/(U+V)`	Sørensen type
`1 - U*V/(U+V-U*V)`	Jaccard type
`1 - sqrt(U*V)`	Ochiai type
`(pmin(U,V) - U*V)/pmin(U,V)`	Simpson type

Function `vegdist` implements Jaccard-type Chao distance, and its documentation contains more complete discussion on the calculation of the terms.

Value

designdist returns an object of class dist.

Note

designdist does not use compiled code, but it is based on vectorized R code. The designdist function can be much faster than vegdist, although the latter uses compiled code. However, designdist cannot skip missing values and uses much more memory during calculations.

The use of sum terms can be numerically unstable. In particularly, when these terms are large, the precision may be lost. The risk is large when the number of columns is high, and particularly large with quadratic terms. For precise calculations it is better to use functions like dist and vegdist which are more robust against numerical problems.

Author(s)

Jari Oksanen

References

Chao, A., Chazdon, R. L., Colwell, R. K. and Shen, T. (2005) A new statistical approach for assessing similarity of species composition with incidence and abundance data. *Ecology Letters* **8**, 148–159.

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

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English ed. Elsevier

See Also

vegdist, betadiver, dist, raupcrick.

Examples

```
data(BCI)
## Five ways of calculating the same Sørensen dissimilarity
d0 <- vegdist(BCI, "bray", binary = TRUE)
d1 <- designdist(BCI, "(A+B-2*J)/(A+B)")
d2 <- designdist(BCI, "(b+c)/(2*a+b+c)", abcd = TRUE)
d3 <- designdist(BCI, "gamma/alpha - 1", alphagamma = TRUE)
d4 <- designdist(BCI, "dist(x, 'manhattan')/(A+B)")
## Zero-adjusted Bray-Curtis of Clarke et al. (J Exp Marine Biol & Ecol
## 330:55-80; 2006)
dbr0 <- designdist(BCI, "(A+B-2*J)/(A+B+2*min(x[x>0]))", terms = "minimum")
## Arrhenius dissimilarity: the value of z in the species-area model
## S = c*A^z when combining two sites of equal areas, where S is the
## number of species, A is the area, and c and z are model parameters.
## The A below is not the area (which cancels out), but number of
## species in one of the sites, as defined in designdist().
dis <- designdist(BCI, "(log(A+B-J)-log(A+B)+log(2))/log(2)")
## This can be used in clustering or ordination...
ordiplot(cmdscale(dis))
```

```
## ... or in analysing beta diversity (without gradients)
summary(dis)
```

---

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

Description

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

Usage

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

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

Arguments

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

Details

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

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

There is little need to call these functions directly. However, they are called implicitly in step function used in automatic selection of constraining variables. You should check the resulting model with some other criteria, because the statistics used here are unfounded. In particular, the

penalty k is not properly defined, and the default k = 2 is not justified theoretically. If you have only continuous covariates, the step function will base the model building on magnitude of eigenvalues, and the value of k only influences the stopping point (but the variables with the highest eigenvalues are not necessarily the most significant in permutation tests in anova.cca). If you also have multi-class factors, the value of k will have a capricious effect in model building. The step function will pass arguments to add1.cca and drop1.cca, and setting test = "permutation" will provide permutation tests of each deletion and addition which can help in judging the validity of the model building.

Value

The deviance functions return "deviance", and extractAIC returns effective degrees of freedom and "AIC".

Note

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

Author(s)

Jari Oksanen

References

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

See Also

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

Examples

```
# The deviance of correspondence analysis equals Chi-square
data(dune)
data(dune.env)
chisq.test(dune)
deviance(cca(dune))
# Stepwise selection (forward from an empty model "dune ~ 1")
ord <- cca(dune ~ ., dune.env)
step(cca(dune ~ 1, dune.env), scope = formula(ord))
```

`dispindmorisita` *Morisita index of intraspecific aggregation*

Description

Calculates the Morisita index of dispersion, standardized index values, and the so called clumpedness and uniform indices.

Usage

```
dispindmorisita(x, unique.rm = FALSE, crit = 0.05, na.rm = FALSE)
```

Arguments

`x`	community data matrix, with sites (samples) as rows and species as columns.
`unique.rm`	logical, if `TRUE`, unique species (occurring in only one sample) are removed from the result.
`crit`	two-sided p-value used to calculate critical Chi-squared values.
`na.rm`	logical. Should missing values (including `NaN`) be omitted from the calculations?

Details

The Morisita index of dispersion is defined as (Morisita 1959, 1962):

`Imor = n * (sum(xi^2) - sum(xi)) / (sum(xi)^2 - sum(xi))`

where xi is the count of individuals in sample i, and n is the number of samples ($i = 1, 2, \ldots, n$). $Imor$ has values from 0 to n. In uniform (hyperdispersed) patterns its value falls between 0 and 1, in clumped patterns it falls between 1 and n. For increasing sample sizes (i.e. joining neighbouring quadrats), $Imor$ goes to n as the quadrat size approaches clump size. For random patterns, $Imor = 1$ and counts in the samples follow Poisson frequency distribution.

The deviation from random expectation (null hypothesis) can be tested using critical values of the Chi-squared distribution with $n - 1$ degrees of freedom. Confidence intervals around 1 can be calculated by the clumped $Mclu$ and uniform $Muni$ indices (Hairston et al. 1971, Krebs 1999) (Chi2Lower and Chi2Upper refers to e.g. 0.025 and 0.975 quantile values of the Chi-squared distribution with $n - 1$ degrees of freedom, respectively, for `crit = 0.05`):

`Mclu = (Chi2Lower - n + sum(xi)) / (sum(xi) - 1)`

`Muni = (Chi2Upper - n + sum(xi)) / (sum(xi) - 1)`

Smith-Gill (1975) proposed scaling of Morisita index from [0, n] interval into [-1, 1], and setting up -0.5 and 0.5 values as confidence limits around random distribution with rescaled value 0. To rescale the Morisita index, one of the following four equations apply to calculate the standardized index $Imst$:

(a) `Imor >= Mclu > 1`: `Imst = 0.5 + 0.5 (Imor - Mclu) / (n - Mclu)`,

(b) `Mclu > Imor >= 1`: `Imst = 0.5 (Imor - 1) / (Mclu - 1)`,

(c) `1 > Imor > Muni`: `Imst = -0.5 (Imor - 1) / (Muni - 1)`,

(d) `1 > Muni > Imor`: `Imst = -0.5 + 0.5 (Imor - Muni) / Muni`.

Value

Returns a data frame with as many rows as the number of columns in the input data, and with four columns. Columns are: `imor` the unstandardized Morisita index, `mclu` the clumpedness index, `muni` the uniform index, `imst` the standardized Morisita index, `pchisq` the Chi-squared based probability for the null hypothesis of random expectation.

Note

A common error found in several papers is that when standardizing as in the case (b), the denominator is given as `Muni - 1`. This results in a hiatus in the [0, 0.5] interval of the standardized index. The root of this typo is the book of Krebs (1999), see the Errata for the book (Page 217, currently https://www.zoology.ubc.ca/~krebs/downloads/errors_2nd_printing.pdf).

Author(s)

Péter Sólymos, <solymos@ualberta.ca>

References

Morisita, M. 1959. Measuring of the dispersion of individuals and analysis of the distributional patterns. *Mem. Fac. Sci. Kyushu Univ. Ser. E* 2, 215–235.

Morisita, M. 1962. Id-index, a measure of dispersion of individuals. *Res. Popul. Ecol.* 4, 1–7.

Smith-Gill, S. J. 1975. Cytophysiological basis of disruptive pigmentary patterns in the leopard frog, *Rana pipiens*. II. Wild type and mutant cell specific patterns. *J. Morphol.* 146, 35–54.

Hairston, N. G., Hill, R. and Ritte, U. 1971. The interpretation of aggregation patterns. In: Patil, G. P., Pileou, E. C. and Waters, W. E. eds. *Statistical Ecology 1: Spatial Patterns and Statistical Distributions*. Penn. State Univ. Press, University Park.

Krebs, C. J. 1999. *Ecological Methodology*. 2nd ed. Benjamin Cummings Publishers.

Examples

```
data(dune)
x <- dispindmorisita(dune)
x
y <- dispindmorisita(dune, unique.rm = TRUE)
y
dim(x) ## with unique species
dim(y) ## unique species removed
```

---

dispweight — *Dispersion-based weighting of species counts*

Description

Transform abundance data downweighting species that are overdispersed to the Poisson error.

Usage

```
dispweight(comm, groups, nsimul = 999, nullmodel = "c0_ind",
    plimit = 0.05)
gdispweight(formula, data, plimit = 0.05)
## S3 method for class 'dispweight'
summary(object, ...)
```

Arguments

comm	Community data matrix.
groups	Factor describing the group structure. If missing, all sites are regarded as belonging to one group. NA values are not allowed.
nsimul	Number of simulations.
nullmodel	The nullmodel used in commsim within groups. The default follows Clarke et al. (2006).
plimit	Downweight species if their p-value is at or below this limit.
formula, data	Formula where the left-hand side is the community data frame and right-hand side gives the explanatory variables. The explanatory variables are found in the data frame given in data or in the parent frame.
object	Result object from dispweight or gdispweight.
...	Other parameters passed to functions.

Details

The dispersion index (D) is calculated as ratio between variance and expected value for each species. If the species abundances follow Poisson distribution, expected dispersion is $E(D) = 1$, and if $D > 1$, the species is overdispersed. The inverse $1/D$ can be used to downweight species abundances. Species are only downweighted when overdispersion is judged to be statistically significant (Clarke et al. 2006).

Function dispweight implements the original procedure of Clarke et al. (2006). Only one factor can be used to group the sites and to find the species means. The significance of overdispersion is assessed freely distributing individuals of each species within factor levels. This is achieved by using nullmodel "c0_ind" (which accords to Clarke et al. 2006), but other nullmodels can be used, though they may not be meaningful (see commsim for alternatives). If a species is absent in some factor level, the whole level is ignored in calculation of overdispersion, and the number of degrees of freedom can vary among species. The reduced number of degrees of freedom is used as a divisor for overdispersion D, and such species have higher dispersion and hence lower weights in transformation.

Function gdispweight is a generalized parametric version of dispweight. The function is based on glm with quasipoisson error family. Any glm model can be used, including several factors or continuous covariates. Function gdispweight uses the same test statistic as dispweight (Pearson Chi-square), but it does not ignore factor levels where species is absent, and the number of degrees of freedom is equal for all species. Therefore transformation weights can be higher than in dispweight. The gdispweight function evaluates the significance of overdispersion parametrically from Chi-square distribution (pchisq).

Functions dispweight and gdispweight transform data, but they add information on overdispersion and weights as attributes of the result. The summary can be used to extract and print that information.

Value

Function returns transformed data with the following new attributes:

D	Dispersion statistic.
df	Degrees of freedom for each species.
p	p-value of the Dispersion statistic D.
weights	weights applied to community data.
nsimul	Number of simulations used to assess the p-value, or NA when simulations were not performed.
nullmodel	The name of commsim null model, or NA when simulations were not performed.

Author(s)

Eduard Szöcs <eduardszoesc@gmail.com> wrote the original dispweight, Jari Oksanen significantly modified the code, provided support functions and developed gdispweight.

References

Clarke, K. R., M. G. Chapman, P. J. Somerfield, and H. R. Needham. 2006. Dispersion-based weighting of species counts in assemblage analyses. *Marine Ecology Progress Series*, 320, 11–27.

Examples

```
data(mite, mite.env)
## dispweight and its summary
mite.dw <- with(mite.env, dispweight(mite, Shrub, nsimul = 99))
## IGNORE_RDIFF_BEGIN
summary(mite.dw)
## IGNORE_RDIFF_END
## generalized dispersion weighting
mite.dw <- gdispweight(mite ~ Shrub + WatrCont, data = mite.env)
rda(mite.dw ~ Shrub + WatrCont, data = mite.env)
```

---

distconnected *Connectedness of Dissimilarities*

Description

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

Usage

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

no.shared(x)
```

Arguments

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

Details

Data sets are disconnected if they have sample plots or groups of sample plots which share no species with other sites or groups of sites. Such data sets cannot be sensibly ordinated by any unconstrained method because these subsets cannot be related to each other. For instance, correspondence analysis will polarize these subsets with eigenvalue 1. Neither can such dissimilarities be transformed with stepacross, because there is no path between all points, and result will contain NAs. Function distconnected will find such subsets in dissimilarity matrices. The function will return a grouping vector that can be used for sub-setting the data. If data are connected, the result vector will be all 1s. The connectedness between two points can be defined either by a threshold toolong or using input dissimilarities with 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 distconnected uses depth-first search (Sedgewick 1990).

Value

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

Author(s)

Jari Oksanen

References

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

See Also

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

Examples

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

---

diversity *Ecological Diversity Indices*

Description

Shannon, Simpson, and Fisher diversity indices and species richness.

Usage

```
diversity(x, index = "shannon", groups, equalize.groups = FALSE,
   MARGIN = 1, base = exp(1))
simpson.unb(x, inverse = FALSE)
fisher.alpha(x, MARGIN = 1, ...)
specnumber(x, groups, MARGIN = 1)
```

Arguments

x	Community data, a matrix-like object or a vector.
index	Diversity index, one of "shannon", "simpson" or "invsimpson".
MARGIN	Margin for which the index is computed.
base	The logarithm base used in shannon.
inverse	Use inverse Simpson similarly as in diversity(x, "invsimpson").
groups	A grouping factor: if given, finds the diversity of communities pooled by the groups.
equalize.groups	Instead of observed abundances, standardize all communities to unit total.
...	Parameters passed to the function.

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 $1/D$.

`simpson.unb` finds unbiased Simpson indices for discrete samples (Hurlbert 1971, eq. 5). These are less sensitive to sample size than the basic Simpson indices. The unbiased indices can be only calculated for data of integer counts.

The `diversity` function can find the total (or gamma) diversity of pooled communities with argument `groups`. The average alpha diversity can be found as the mean of diversities by the same groups, and their difference or ratio is an estimate of beta diversity (see Examples). The pooling can be based either on the observed abundances, or all communities can be equalized to unit total before pooling; see Jost (2007) for discussion. Functions `adipart` and `multipart` provide canned alternatives for estimating alpha, beta and gamma diversities in hierarchical settings.

`fisher.alpha` estimates the α parameter of Fisher's logarithmic series (see `fisherfit`). The estimation is possible only for genuine counts of individuals.

None of these diversity indices is usable for empty sampling units without any species, but some of the indices can give a numeric value. Filtering out these cases is left for the user.

Function `specnumber` finds the number of species. With `MARGIN = 2`, it finds frequencies of species. If `groups` is given, finds the total number of species in each group (see example on finding one kind of beta diversity with this option).

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

Value

A vector of diversity indices or numbers of species.

Author(s)

Jari Oksanen and Bob O'Hara (`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.

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

Jost, L. (2007) Partitioning diversity into independent alpha and beta components. *Ecology* **88**, 2427–2439.

See Also

These functions calculate only some basic indices, but many others can be derived with them (see Examples). Facilities related to diversity are discussed in a **vegan** vignette that can be read with `browseVignettes("vegan")`. Functions `renyi` and `tsallis` estimate a series of generalized diversity indices. Function `rarefy` finds estimated number of species for given sample size. Beta diversity can be estimated with `betadiver`. Diversities can be partitioned with `adipart` and `multipart`.

Examples

```
data(BCI, BCI.env)
H <- diversity(BCI)
simp <- diversity(BCI, "simpson")
invsimp <- diversity(BCI, "inv")
## Unbiased Simpson
unbias.simp <- simpson.unb(BCI)
## Fisher alpha
alpha <- fisher.alpha(BCI)
## Plot all
pairs(cbind(H, simp, invsimp, unbias.simp, 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)
## beta diversity defined as gamma/alpha - 1:
## alpha is the average no. of species in a group, and gamma is the
## total number of species in the group
(alpha <- with(BCI.env, tapply(specnumber(BCI), Habitat, mean)))
(gamma <- with(BCI.env, specnumber(BCI, Habitat)))
gamma/alpha - 1
## similar calculations with Shannon diversity
(alpha <- with(BCI.env, tapply(diversity(BCI), Habitat, mean))) # average
(gamma <- with(BCI.env, diversity(BCI, groups=Habitat))) # pooled
## additive beta diversity based on Shannon index
gamma-alpha
```

---

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

dune is a data frame of observations of 30 species at 20 sites. The species names are abbreviated to 4+4 letters (see make.cepnames). The following names are changed from the original source (Jongman et al. 1987): *Leontodon autumnalis* to *Scorzoneroides*, and *Potentilla palustris* to *Comarum*.

dune.env is a data frame of 20 observations on the following 5 variables:

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

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

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

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

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

Source

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

Examples

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

---

dune.taxon *Taxonomic Classification and Phylogeny of Dune Meadow Species*

Description

Classification table of the species in the dune data set.

Usage

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

Format

dune.taxon is data frame with 30 species (rows) classified into five taxonomic levels (columns). dune.phylodis is a dist object of estimated coalescence ages extracted from doi:10.5061/dryad.63q27 (Zanne et al. 2014) using tools in packages **ape** and **phylobase**.

Details

The families and orders are based on APG IV (2016) in vascular plants and on Hill et al. (2006) in mosses. The higher levels (superorder and subclass) are based on Chase & Reveal (2009). Chase & Reveal (2009) treat Angiosperms and mosses as subclasses of class Equisetopsida (land plants), but brylogists have traditionally used much more inflated levels which are adjusted here to match Angiosperm classification.

References

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

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

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

Zanne A.E., Tank D.C., Cornwell, W.K., Eastman J.M., Smith, S.A., FitzJohn, R.G., McGlinn, D.J., O'Meara, B.C., Moles, A.T., Reich, P.B., Royer, D.L., Soltis, D.E., Stevens, P.F., Westoby, M., Wright, I.J., Aarssen, L., Bertin, R.I., Calaminus, A., Govaerts, R., Hemmings, F., Leishman, M.R., Oleksyn, J., Soltis, P.S., Swenson, N.G., Warman, L. & Beaulieu, J.M. (2014) Three keys to the radiation of angiosperms into freezing environments. *Nature* **506**: 89–92.

See Also

Functions `taxondive`, `treedive`, and `treedist` use these data sets.

Examples

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

`eigenvals` *Extract Eigenvalues from an Ordination Object*

Description

Function extracts eigenvalues from an object that has them. Many multivariate methods return such objects.

Usage

```
eigenvals(x, ...)
## S3 method for class 'cca'
eigenvals(x, model = c("all", "unconstrained", "constrained"),
          constrained = NULL, ...)
## S3 method for class 'decorana'
eigenvals(x, kind = c("additive", "axiswise", "decorana"),
           ...)
## S3 method for class 'eigenvals'
summary(object, ...)
```

Arguments

x	An object from which to extract eigenvalues.
object	An eigenvals result object.
model	Which eigenvalues to return for objects that inherit from class "cca" only.
constrained	Return only constrained eigenvalues. Deprecated as of vegan 2.5-0. Use model instead.
kind	Kind of eigenvalues returned for decorana. Only "additive" eigenvalues can be used for reporting importances of components in summary. "axiswise" gives the non-additive eigenvalues, and "decorana" the decorana values (see decorana for details).
...	Other arguments to the functions (usually ignored)

Details

This is a generic function that has methods for cca, wcmdscale, pcnm, prcomp, princomp, dudi (of **ade4**), and pca and pco (of **labdsv**) result objects. The default method also extracts eigenvalues if the result looks like being from eigen or svd. Functions prcomp and princomp contain square roots of eigenvalues that all called standard deviations, but eigenvals function returns their squares. Function svd contains singular values, but function eigenvals returns their squares. For constrained ordination methods cca, rda and capscale the function returns the both constrained and unconstrained eigenvalues concatenated in one vector, but the partial component will be ignored. However, with argument constrained = TRUE only constrained eigenvalues are returned.

The summary of eigenvals result returns eigenvalues, proportion explained and cumulative proportion explained. The result object can have some negative eigenvalues (wcmdscale, dbrda, pcnm) which correspond to imaginary axes of Euclidean mapping of non-Euclidean distances (Gower 1985). In these case real axes (corresponding to positive eigenvalues) will "explain" proportion >1 of total variation, and negative eigenvalues bring the cumulative proportion to 1. capscale will only find the positive eigenvalues and only these are used in finding proportions. For decorana the importances and cumulative proportions are only reported for kind = "additive", because other alternatives do not add up to total inertia of the input data.

Value

An object of class "eigenvals", which is a vector of eigenvalues.

The summary method returns an object of class "summary.eigenvals", which is a matrix.

Author(s)

Jari Oksanen.

References

Gower, J. C. (1985). Properties of Euclidean and non-Euclidean distance matrices. *Linear Algebra and its Applications* 67, 81–97.

See Also

eigen, svd, prcomp, princomp, cca, rda, capscale, wcmdscale, cca.object.

Examples

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

## choose which eignevalues to return
eigenvals(mod, model = "unconstrained")
```

---

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

Description

The function fits environmental vectors or factors onto an ordination. The projections of points onto vectors have maximum correlation with corresponding environmental variables, and the factors show the averages of factor levels. For continuous variables this is equal to fitting a linear trend surface (plane in 2D) for a variable (see ordisurf); this trend surface can be presented by showing its gradient (direction of steepest increase) using an arrow. The environmental variables are the dependent variables that are explained by the ordination scores, and each dependent variable is analysed separately.

Usage

```
## Default S3 method:
envfit(ord, env, permutations = 999, strata = NULL,
   choices=c(1,2),  display = "sites", w, na.rm = FALSE, ...)
## S3 method for class 'formula'
envfit(formula, data, ...)
## S3 method for class 'envfit'
plot(x, choices = c(1,2), labels, arrow.mul, at = c(0,0),
   axis = FALSE, p.max = NULL, r2.min = NULL, col = "blue", bg,
   optimize = FALSE, cex = 1, add = TRUE, ...)
## S3 method for class 'envfit'
scores(x, display = c("vectors", "factors"), choices,
   arrow.mul=1, tidy = FALSE, ...)
## S3 replacement method for class 'envfit'
names(x) <- value
vectorfit(X, P, permutations = 0, strata = NULL, w, ...)
factorfit(X, P, permutations = 0, strata = NULL, w, ...)
```

Arguments

ord	An ordination object or other structure from which the ordination scores can be extracted (including a data frame or matrix of scores).

`env`	Data frame, matrix or vector of environmental variables. The variables can be of mixed type (factors, continuous variables) in data frames.
`X`	Matrix or data frame of ordination scores.
`P`	Data frame, matrix or vector of environmental variable(s). These must be continuous for `vectorfit` and factors or characters for `factorfit`.
`permutations`	a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices. Set `permutations = 0` to skip permutations.
`formula, data`	Model `formula` and data.
`na.rm`	Remove points with missing values in ordination scores or environmental variables. The operation is casewise: the whole row of data is removed if there is a missing value and `na.rm = TRUE`.
`x`	A result object from `envfit`. For `ordiArrowMul` and `ordiArrowTextXY` this must be a two-column matrix (or matrix-like object) containing the coordinates of arrow heads on the two plot axes, and other methods extract such a structure from the `envfit` results.
`choices`	Axes to plotted.
`tidy`	Return scores that are compatible with **ggvegan**: all scores are in a single `data.frame`, score type is identified by factor variable `type` (`"Vector"` or `"Centroid"`), the names by variable `label`. These scores are incompatible with conventional `plot` functions, but they can be used in **ggvegan** and **ggplot2**.
`labels`	Change plotting labels. The argument should be a list with elements `vectors` and `factors` which give the new plotting labels. If either of these elements is omitted or `NULL`, the default labels will be used. If there is only one type of elements (only `vectors` or only `factors`), the labels can be given as vector. The default labels can be displayed with `labels` command. The argument changes names only in the `plot` and the fitted object is unchanged; use `names` to change the names in the object.
`value`	Names of environmental variables used to replace the names in the result object. This should be a list with items `factors` and `vectors`, or it can be vector of names if only factors or vectors were calculated. If either `factors` or `vectors` is omitted or `NULL`, names will not be replaced for that item. The replacement function `names` will permanently change the result object; use argument `labels` in `plot` to change names in the graph and keep the result object unchanged.
`arrow.mul`	Multiplier for vector lengths. The arrows are automatically scaled similarly as in `plot.cca` if this is not given in `plot` and `add = TRUE`. However, in `scores` it can be used to adjust arrow lengths when the `plot` function is not used.
`at`	The origin of fitted arrows in the plot.
`axis`	Add axis of the scaling of fitted arrows in units of correlation on upper and right axes.
`p.max, r2.min`	Maximum estimated P value and minimum r^2 for displayed variables. You must calculate P values with setting `permutations` to use `p.max`.
`col`	Colour in plotting.

bg	Background colour for labels. If bg is set, the labels are displayed with ordilabel instead of text. See Examples for using semitransparent background.
optimize	Optimize the location of text labels to minimize overlap. If TRUE, factors will be shown with a cross in their exact location, and text for factors and vectors will be optimized.
cex	Character expansion of text labels.
add	Add results to an existing ordination plot or generate a new 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, dbrda). 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 with 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.

The ordination results are extracted with scores and all extra arguments are passed to the scores. The fitted models only apply to the results defined when extracting the scores when using envfit. For instance, scaling in constrained ordination (see scores.rda, scores.cca) must be set in the same way in envfit and in the plot or the ordination results (see Examples).

The printed output of continuous variables (vectors) gives the direction cosines which are the coordinates of the heads of unit length vectors. In plot these are scaled by their correlation (square root of the column r2) so that "weak" predictors have shorter arrows than "strong" predictors. You can see the scaled relative lengths using command scores. The plotted (and scaled) arrows are further adjusted to the current graph using a constant multiplier: this will keep the relative r2-scaled lengths of the arrows but tries to fill the current plot. You can see the multiplier using ordiArrowMul(result_of_envfit), and set it with the argument arrow.mul.

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. See permutations for additional details on permutation tests in Vegan.

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

The lengths of arrows for fitted vectors are automatically adjusted for the physical size of the plot, and the arrow lengths cannot be compared across plots. For similar scaling of arrows, you must explicitly set the `arrow.mul` argument in the `plot` command; see `ordiArrowMul`.

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, or a named list of both.

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.
`control`	A list of control values for the permutations as returned by the function `how`.
`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` and Redundancy analysis (`rda`), 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, varechem)
ord <- metaMDS(varespec)
(fit <- envfit(ord, varechem, perm = 999))
scores(fit, "vectors")
plot(ord)
plot(fit)
plot(fit, p.max = 0.05, col = "red")
## Interpretation of fitted arrow: assume linear trend surface, and
## the arrow shows the direction of steepest increase, and fitted
## values are found by projecting points onto the arrow
plot(ord, type = "p")
plot(envfit(ord ~ Al, varechem, permutations = 0))
ordisurf(ord ~ Al, varechem, add = TRUE, knots = 1)
title(main = "Interpretation of the arrow")
## 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 = 999, display = "lc")
plot(fit, p.max = 0.05, col = "red")
## 'scaling' must be set similarly in envfit and in ordination plot
plot(ord, type = "p", scaling = "sites")
fit <- envfit(ord, varechem, perm = 0, display = "lc", scaling = "sites")
plot(fit, col = "red")

## Class variables, formula interface, and displaying the
## inter-class variability with ordispider, and semitransparent
## white background for labels (semitransparent colours are not
## supported by all graphics devices)
data(dune)
data(dune.env)
ord <- cca(dune)
fit <- envfit(ord ~ Moisture + A1, dune.env, perm = 0)
plot(ord, type = "n")
with(dune.env, ordispider(ord, Moisture, col="skyblue"))
with(dune.env, points(ord, display = "sites", col = as.numeric(Moisture),
                      pch=16))
plot(fit, cex=1.2, axis=TRUE, bg = rgb(1, 1, 1, 0.5))
## Use shorter labels for factor centroids
labels(fit)
plot(ord)
plot(fit, labels=list(factors = paste0("M", c(1,2,4,5))),
   bg = rgb(1,1,0,0.5))
## change names permanently
names(fit) <- list(factors = paste0("M", c(1,2,4,5)))
fit
```

`eventstar` *Scale Parameter at the Minimum of the Tsallis Evenness Profile*

Description

The function `eventstar` finds the minimum (q^*) of the evenness profile based on the Tsallis entropy. This scale factor of the entropy represents a specific weighting of species relative frequencies that leads to minimum evenness of the community (Mendes et al. 2008).

Usage

```
eventstar(x, qmax = 5)
```

Arguments

`x`	A community matrix or a numeric vector.
`qmax`	Maximum scale parameter of the Tsallis entropy to be used in finding the minimum of Tsallis based evenness in the range `c(0, qmax)`.

Details

The function `eventstar` finds a characteristic value of the scale parameter q of the Tsallis entropy corresponding to minimum of the evenness (equitability) profile based on Tsallis entropy. This value was proposed by Mendes et al. (2008) as q^*.

The q^* index represents the scale parameter of the one parameter Tsallis diversity family that leads to the greatest deviation from the maximum equitability given the relative abundance vector of a community.

The value of q^* is found by identifying the minimum of the evenness profile over scaling factor q by one-dimensional minimization. Because evenness profile is known to be a convex function, it is guaranteed that underlying `optimize` function will find a unique solution if it is in the range `c(0, qmax)`.

The scale parameter value q^* is used to find corresponding values of diversity (H_{q^*}), evenness ($H_{q^*}(\max)$), and numbers equivalent (D_{q^*}). For calculation details, see `tsallis` and Examples below.

Mendes et al. (2008) advocated the use of q^* and corresponding diversity, evenness, and Hill numbers, because it is a unique value representing the diversity profile, and is is positively associated with rare species in the community, thus it is a potentially useful indicator of certain relative abundance distributions of the communities.

Value

A data frame with columns:

`qstar`	scale parameter value $q*$ corresponding to minimum value of Tsallis based evenness profile.
`Estar`	Value of evenness based on normalized Tsallis entropy at q^*.
`Hstar`	Value of Tsallis entropy at q^*.
`Dstar`	Value of Tsallis entropy at q^* converted to numbers equivalents (also called as Hill numbers, effective number of species, 'true' diversity; cf. Jost 2007).

See `tsallis` for calculation details.

Note

Values for q^* found by Mendes et al. (2008) ranged from 0.56 and 1.12 presenting low variability, so an interval between 0 and 5 should safely encompass the possibly expected q^* values in practice, but profiling the evenness and changing the value of the `qmax` argument is advised if output values near the range limits are found.

Author(s)

Eduardo Ribeiro Cunha <edurcunha@gmail.com> and Heloisa Beatriz Antoniazi Evangelista <helobeatriz@gmail.com>, with technical input of Péter Sólymos.

References

Mendes, R.S., Evangelista, L.R., Thomaz, S.M., Agostinho, A.A. and Gomes, L.C. (2008) A unified index to measure ecological diversity and species rarity. *Ecography* **31**, 450–456.

Jost, L. (2007) Partitioning diversity into independent alpha and beta components. *Ecology* **88**, 2427–2439.

Tsallis, C. (1988) Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phis.* **52**, 479–487.

See Also

Tsallis entropy: `tsallis`

Examples

```
data(BCI)
(x <- eventstar(BCI[1:5,]))
## profiling
y <- as.numeric(BCI[10,])
(z <- eventstar(y))
q <- seq(0, 2, 0.05)
Eprof <- tsallis(y, scales=q, norm=TRUE)
Hprof <- tsallis(y, scales=q)
Dprof <- tsallis(y, scales=q, hill=TRUE)
opar <- par(mfrow=c(3,1))
plot(q, Eprof, type="l", main="Evenness")
abline(v=z$qstar, h=tsallis(y, scales=z$qstar, norm=TRUE), col=2)
plot(q, Hprof, type="l", main="Diversity")
abline(v=z$qstar, h=tsallis(y, scales=z$qstar), col=2)
plot(q, Dprof, type="l", main="Effective number of species")
abline(v=z$qstar, h=tsallis(y, scales=z$qstar, hill=TRUE), col=2)
par(opar)
```

`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, ...)
prestonfit(x, tiesplit = TRUE, ...)
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, ...)
## S3 method for class 'fisher'
plot(x, xlab = "Frequency", ylab = "Species", bar.col = "skyblue",
             kind = c("bar", "hiplot", "points", "lines"), add = FALSE, ...)
as.preston(x, tiesplit = TRUE, ...)
## S3 method for class 'preston'
plot(x, xlab = "Frequency", ylab = "Species", bar.col = "skyblue", ...)
## S3 method for class 'preston'
lines(x, xadjust = 0.5, ...)
```

Arguments

`x`	Community data vector for fitting functions or their result object for `plot` functions.
`tiesplit`	Split frequencies $1, 2, 4, 8$ etc between adjacent octaves.
`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.
`kind`	Kind of plot to drawn: `"bar"` is similar bar plot as in `plot.fisherfit`, `"hiplot"` draws vertical lines as with `plot(..., type="h")`, and `"points"` and `"lines"` are obvious.

add	Add to an existing plot.
xadjust	Adjustment of horizontal positions in octaves.
...	Other parameters passed to functions. Ignored in prestonfit and tiesplit passed to as.preston in prestondistr.

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 is possible only for genuine counts of individuals. The parameter α is used as a diversity index which 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. Diversity will be given as NA for communities with one (or zero) species: there is no reliable way of estimating their diversity, even if the equations will return a bogus numeric value in some cases.

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. It seems that Preston regarded frequencies 1, 2, 4, *etc.*. as "tied" between octaves (Williamson & Gaston 2005). This means that only half of the species with frequency 1 are shown in the lowest octave, and the rest are transferred to the second octave. Half of the species from the second octave are transferred to the higher one as well, but this is usually not as large a number of species. This practise makes data look more lognormal by reducing the usually high lowest octaves. This can be achieved by setting argument tiesplit = TRUE. With tiesplit = FALSE the frequencies are not split, but all ones are in the lowest octave, all twos in the second, etc. Williamson & Gaston (2005) discuss alternative definitions in detail, and they should be consulted for a critical review of log-Normal model.

Any logseries data will look like lognormal when plotted in Preston's 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 (when tiesplit = FALSE) or quasi-Poisson (when tiesplit = TRUE) 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. Function as.preston transforms abundance data to octaves. Argument tiesplit will not influence the fit in prestondistr, but it will influence the barplot of the octaves.

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

Value

The function `prestonfit` returns an object with fitted `coefficients`, and with observed (`freq`) and fitted (`fitted`) frequencies, and a string describing the fitting `method`. Function `prestondistr` omits the entry `fitted`. The function `fisherfit` returns the result of `nlm`, where item `estimate` is α. The result object is amended with the `nuisance` parameter and item `fisher` for the observed data from `as.fisher`

Author(s)

Bob O'Hara 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.

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

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

See Also

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

Examples

```
data(BCI)
mod <- fisherfit(BCI[5,])
mod
# 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, choices, display = c("species", "sites"),
    model = c("CCA", "CA"), summarize = FALSE, addprevious = FALSE, ...)
inertcomp(object, display = c("species", "sites"),
    unity = FALSE, proportional = FALSE)
spenvcor(object)
intersetcor(object)
vif.cca(object)
## S3 method for class 'cca'
alias(object, names.only = FALSE, ...)
```

Arguments

object	A result object from cca, rda, dbrda or capscale.
display	Display "species" or "sites". Species are not available in dbrda and capscale.
choices	Axes shown. Default is to show all axes of the "model".
model	Show constrained ("CCA") or unconstrained ("CA") results.
summarize	Show only the accumulated total.
addprevious	Add the variation explained by previous components when statistic="explained". For model = "CCA" add conditioned (partialled out) variation, and for model = "CA" add both conditioned and constrained variation. This will give cumulative explanation with previous components.
unity	Scale inertia components to unit sum (sum of all items is 1).
proportional	Give the inertia components as proportional for the corresponding total of the item (sum of each row is 1). This option takes precedence over unity.
names.only	Return only names of aliased variable(s) instead of defining equations.
...	Other parameters to the functions.

Details

Function goodness gives cumulative proportion of inertia accounted by species up to chosen axes. The proportions can be assessed either by species or by sites depending on the argument display, but species are not available in distance-based dbrda. The function is not implemented for capscale.

Function inertcomp decomposes the inertia into partial, constrained and unconstrained components for each site or species. Legendre & De Cáceres (2012) called these inertia components as local contributions to beta-diversity (LCBD) and species contributions to beta-diversity (SCBD), and they give these as relative contributions summing up to unity (argument unity = TRUE). For this interpretation, appropriate dissimilarity measures should be used in dbrda or appropriate standardization in rda (Legendre & De Cáceres 2012). The function is not implemented for capscale.

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

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

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

Value

The functions return matrices or vectors as is appropriate.

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.

Legendre, P. & De Cáceres, M. (2012). Beta diversity as the variance of community data: dissimilarity coefficients and partitioning. *Ecology Letters* 16, 951–963. doi:10.1111/ele.12141

See Also

cca, rda, dbrda, capscale.

Examples

```
data(dune, dune.env)
mod <- cca(dune ~ A1 + Management + Condition(Moisture), data=dune.env)
goodness(mod, addprevious = TRUE)
goodness(mod, addprevious = TRUE, summ = TRUE)
## Fit of species in 2 dimensions
good <- goodness(mod, choices = 1:2, summarize = TRUE)
sort(good)
## Drop poorly fitting species from a plot
plot(mod, spe.par = list(select = good > 0.1, optimize = TRUE,
   bg = "yellow"))
## Inertia components
inertcomp(mod, prop = TRUE)
inertcomp(mod)
## vif.cca
vif.cca(mod)
## Aliased constraints
mod <- cca(dune ~ ., dune.env)
mod
vif.cca(mod)
alias(mod)
with(dune.env, table(Management, Manure))
## The standard correlations (not recommended)
## IGNORE_RDIFF_BEGIN
spenvcor(mod)
intersetcor(mod)
## IGNORE_RDIFF_END
```

---

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, ...)
## Default S3 method:
stressplot(object, dis, pch, p.col = "blue", l.col = "red",
    lwd = 2, ...)
```

Arguments

`object`	A result object from `metaMDS`, `monoMDS` or `isoMDS`.
`dis`	Dissimilarities. This should not be used with `metaMDS` or `monoMDS`, but must be used with when the dissimilarities cannot be reconstructed from the result object.
`pch`	Plotting character for points. Default is dependent on the number of points.
`p.col`, `l.col`	Point and line colours.
`lwd`	Line width. For `monoMDS` the default is `lwd = 1` if more than two lines are drawn, and `lwd = 2` otherwise.
`...`	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` draws a Shepard diagram which is a plot of ordination distances and monotone or linear fit line against original dissimilarities. In addition, it displays two correlation-like statistics on the goodness of fit in the graph. The nonmetric fit is based on stress S and defined as $R^2 = 1 - S^2$. The "linear fit" is the squared correlation between fitted values and ordination distances. For `monoMDS`, the "linear fit" and R^2 from "stress type 2" are equal.

Both functions can be used with `metaMDS`, `monoMDS` and `isoMDS`. The original dissimilarities should not be given for `monoMDS` or `metaMDS` results, but they must given if the result object has no information to reconstruct dissmilarities. The functions checks that dissimilarities are consistent with current ordination, and refuses to analyse inconsistent dissimilarities. Function `goodness.metaMDS` is generic in **vegan**, but you must spell its name completely if the result has no `class`.

Value

Function `goodness` returns a vector of values. Function `stressplot` returns invisibly an object with items for original dissimilarities, ordination distances and fitted values.

Author(s)

Jari Oksanen.

See Also

`metaMDS`, `monoMDS`, `isoMDS`, `Shepard`. Similar diagrams for eigenvector ordinations can be drawn with `stressplot.wcmdscale`, `stressplot.cca`.

Examples

```
data(varespec)
mod <- metaMDS(varespec)
stressplot(mod)
gof <- goodness(mod)
gof
```

```
plot(mod, display = "sites", type = "n")
points(mod, display = "sites", cex = 2*gof/mean(gof))
```

---

indpower *Indicator Power of Species*

Description

Indicator power calculation of Halme et al. (2009) or the congruence between indicator and target species.

Usage

```
indpower(x, type = 0)
```

Arguments

x	Community data frame or matrix.
type	The type of statistic to be returned. See Details for explanation.

Details

Halme et al. (2009) described an index of indicator power defined as $IP_I = \sqrt{a \times b}$, where $a = S/O_I$ and $b = 1 - (O_T - S)/(N - O_I)$. N is the number of sites, S is the number of shared occurrences of the indicator (I) and the target (T) species. O_I and O_T are number of occurrences of the indicator and target species. The `type` argument in the function call enables to choose which statistic to return. `type = 0` returns IP_I, `type = 1` returns a, `type = 2` returns b. Total indicator power (TIP) of an indicator species is the column mean (without its own value, see examples). Halme et al. (2009) explain how to calculate confidence intervals for these statistics, see Examples.

Value

A matrix with indicator species as rows and target species as columns (this is indicated by the first letters of the row/column names).

Author(s)

Peter Solymos

References

Halme, P., Mönkkönen, M., Kotiaho, J. S, Ylisirniö, A-L. 2009. Quantifying the indicator power of an indicator species. *Conservation Biology* 23: 1008–1016.

Examples

```
data(dune)
## IP values
ip <- indpower(dune)
## and TIP values
diag(ip) <- NA
(TIP <- rowMeans(ip, na.rm=TRUE))

## p value calculation for a species
## from Halme et al. 2009
## i is ID for the species
i <- 1
fun <- function(x, i) indpower(x)[i,-i]
## 'c0' randomizes species occurrences
os <- oecosimu(dune, fun, "c0", i=i, nsimul=99)
## get z values from oecosimu output
z <- os$oecosimu$z
## p-value
(p <- sum(z) / sqrt(length(z)))
## 'heterogeneity' measure
(chi2 <- sum((z - mean(z))^2))
pchisq(chi2, df=length(z)-1)
## Halme et al.'s suggested output
out <- c(TIP=TIP[i],
    significance=p,
    heterogeneity=chi2,
    minIP=min(fun(dune, i=i)),
    varIP=sd(fun(dune, i=i)^2))
out
```

`influence.cca` *Linear Model Diagnostics for Constrained Ordination*

Description

This set of function extracts influence statistics and some other linear model statistics directly from a constrained ordination result object from `cca`, `rda`, `capscale` or `dbrda`. The constraints are linear model functions and these support functions return identical results as the corresponding linear models (`lm`), and you can use their documentation. The main functions for normal usage are leverage values (`hatvalues`), standardized residuals (`rstandard`), studentized or leave-one-out residuals (`rstudent`), and Cook's distance (`cooks.distance`). Function `influence` returns all these. In addition, `vcov` returns the variance-covariance matrix of coefficients, and its diagonal values the variances of coefficients. Other functions are mainly support functions for these, but they can be used directly.

Usage

```
## S3 method for class 'cca'
influence(model, type = c("response", "canoco"),
```

```
    addFit = FALSE, tidy = FALSE, ...)

## S3 method for class 'cca'
hatvalues(model, ...)
## S3 method for class 'cca'
rstandard(model, type = c("response", "canoco"), ...)
## S3 method for class 'cca'
rstudent(model, type = c("response", "canoco"), ...)
## S3 method for class 'cca'
cooks.distance(model, type = c("response", "canoco"), ...)

## S3 method for class 'cca'
sigma(object, type = c("response", "canoco"), ...)
## S3 method for class 'cca'
SSD(object, type = c("response", "canoco"), ...)
## S3 method for class 'cca'
estVar(object, type = c("response", "canoco"), ...)
## S3 method for class 'cca'
vcov(object, type = c("response", "canoco"), ...)

## S3 method for class 'cca'
qr(x, ...)
## S3 method for class 'cca'
df.residual(object, ...)
```

Arguments

`model`, `object`, `x`	A constrained ordination result object.
`type`	Type of statistics used for extracting raw residuals and residual standard deviation (`sigma`). Either `"response"` for species data or difference of WA and LC scores for `"canoco"`.
`addFit`	Add fit in working scale to the result when `type="response"` (this is impossible with `type="canoco"`).
`tidy`	Return a data frame suitable for **ggplot2** or **lattice**. Each statistic is in one long vector with added columns `site` and `species` or `axis` depending on `type`
.	
`...`	Other arguments to functions (ignored).

Details

The **vegan** algorithm for constrained ordination uses linear model (or weighted linear model in `cca`) to find the fitted values of dependent community data, and constrained ordination is based on this fitted response (Legendre & Legendre 2012). The `hatvalues` give the leverage values of these constraints, and the leverage is independent on the response data. Other influence statistics (`rstandard`, `rstudent`, `cooks.distance`) are based on leverage, and on the raw residuals and residual standard

deviation (sigma). With type = "response" the raw residuals are given by the unconstrained component of the constrained ordination, and influence statistics are a matrix with dimensions no. of observations times no. of species. For cca the statistics are the same as obtained from the lm model using Chi-square standardized species data (see decostand) as dependent variable, and row sums of community data as weights, and for rda the lm model uses non-modified community data and no weights.

The algorithm in the CANOCO software constraints the results during iteration by performing a linear regression of weighted averages (WA) scores on constraints and taking the fitted values of this regression as linear combination (LC) scores (ter Braak 1984). The WA scores are directly found from species scores, but LC scores are linear combinations of constraints in the regression. With type = "canoco" the raw residuals are the differences of WA and LC scores, and the residual standard deviation (sigma) is taken to be the axis sum of squared WA scores minus one. These quantities have no relationship to residual component of ordination, but they rather show the influence of each site on axes. The result is a matrix with dimensions no. of observations times no. of constrained axes.

Function influence returns either a list of the matrices hatvalues, rstandard, rstudent and cooks.distance, or a data frame where each matrix is single variable, with new variables site and species or axis (depending on type) identifying the original rows and columns of matrices. influence can also add fitted values in working scale as a variable. Such data can be used as input for **ggplot2** or **lattice**.

Function vcov returns the matrix of variances and covariances of regression coefficients. The diagonal values of this matrix are the variances, and their square roots give the standard errors of regression coefficients. The function is based on SSD that extracts the sum of squares and crossproducts of residuals. The residuals are defined similarly as in influence measures and with each type they have similar properties and limitations, and define the dimensions of the result matrix.

Author(s)

Jari Oksanen

References

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English ed. Elsevier.

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*.

See Also

Similar linear mmodel functions influence.measures, hatvalues, rstandard, rstudent, cooks.distance, sigma, SSD, estVar, vcov, df.residual.

Examples

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

```
plot(hatvalues(mod), type = "h")
## ordination plot with leverages: points with high leverage have
## similar LC and WA scores
plot(mod, type = "n")
ordispider(mod)       # segment from LC to WA scores
points(mod, dis="si", cex=5*hatvalues(mod), pch=21, bg=2) # WA scores
text(mod, dis="bp", col=4)

## deviation and influence
head(rstandard(mod))
head(cooks.distance(mod))
tabasco(cooks.distance(mod), site.ind = order(hatvalues(mod)))
tabasco(cooks.distance(mod, type="canoco"), site.ind = order(hatvalues(mod)))

## Influence measures from lm
y <- decostand(varespec, "chi.square") # needed in cca
y1 <- with(y, Cladstel)         # take one species for lm
lmod1 <- lm(y1 ~ Al + P + K, varechem, weights = rowSums(varespec))
## numerically identical within 2e-15
all(abs(cooks.distance(lmod1) - cooks.distance(mod)[, "Cladstel"]) < 1e-8)

## t-values of regression coefficients based on type = "canoco"
## residuals
coef(mod)
tval <- coef(mod)/sqrt(diag(vcov(mod, type = "canoco")))
tval
## P-values
zapsmall(2 * pt(abs(tval), df.residual(mod), lower.tail=FALSE))
plot(mod, scaling = "sites", display = c("lc", "reg"))
title(main = "Linear Combination and Regression Scores")
```

`isomap` *Isometric Feature Mapping Ordination*

Description

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

Usage

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

Arguments

dist	Dissimilarities.
ndim	Number of axes in metric scaling (argument k in cmdscale).
epsilon	Shortest dissimilarity retained.
k	Number of shortest dissimilarities retained for a point. If both epsilon and k are given, epsilon will be used.
path	Method used in stepacross to estimate the shortest path, with alternatives "shortest" and "extended".
fragmentedOK	What to do if dissimilarity matrix is fragmented. If TRUE, analyse the largest connected group, otherwise stop with error.
x, object	An isomap result object.
net	Draw the net of retained dissimilarities.
n.col	Colour of drawn net segments. This can also be a vector that is recycled for points, and the colour of the net segment is a mixture of joined points.
type	Plot observations either as "points", "text" or use "none" to plot no observations. The "text" will use ordilabel if net = TRUE and ordiplot if net = FALSE, and pass extra arguments to these functions.
...	Other parameters passed to functions.

Details

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

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

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

The plot function uses internally ordiplot, except that it adds text over net using ordilabel. The plot function passes extra arguments to these functions. In addition, **vegan3d** package has function rgl.isomap to make dynamic 3D plots that can be rotated on the screen.

Value

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

Note

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

Author(s)

Jari Oksanen

References

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

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

See Also

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

Examples

```
## The following examples also overlay minimum spanning tree to
## the graphics in red.
op <- par(mar=c(4,4,1,1)+0.2, mfrow=c(2,2))
data(BCI)
dis <- vegdist(BCI)
tr <- spantree(dis)
pl <- ordiplot(cmdscale(dis), main="cmdscale")
lines(tr, pl, col="red")
ord <- isomap(dis, k=3)
ord
pl <- plot(ord, main="isomap k=3")
lines(tr, pl, col="red")
pl <- plot(isomap(dis, k=5), main="isomap k=5")
lines(tr, pl, col="red")
pl <- plot(isomap(dis, epsilon=0.45), main="isomap epsilon=0.45")
lines(tr, pl, col="red")
par(op)
## colour points and web by the dominant species
dom <- apply(BCI, 1, which.max)
## need nine colours, but default palette  has only eight
op <- palette(c(palette("default"), "sienna"))
plot(ord, pch = 16, col = dom, n.col = dom)
palette(op)
```

kendall.global *Kendall coefficient of concordance*

Description

Function `kendall.global` computes and tests the coefficient of concordance among several judges (variables, species) through a permutation test.

Function `kendall.post` carries out *a posteriori* tests of the contributions of individual judges (variables, species) to the overall concordance of their group through permutation tests.

If several groups of judges are identified in the data table, coefficients of concordance (`kendall.global`) or a posteriori tests (`kendall.post`) will be computed for each group separately. Use in ecology: to identify significant species associations.

Usage

```
kendall.global(Y, group, nperm = 999, mult = "holm")
kendall.post(Y, group, nperm = 999, mult = "holm")
```

Arguments

`Y`	Data file (data frame or matrix) containing quantitative or semiquantitative data. Rows are objects and columns are judges (variables). In community ecology, that table is often a site-by-species table.
`group`	A vector defining how judges should be divided into groups. See example below. If groups are not explicitly defined, all judges in the data file will be considered as forming a single group.
`nperm`	Number of permutations to be performed. Default is 999.
`mult`	Correct P-values for multiple testing using the alternatives described in `p.adjust` and in addition `"sidak"` (see Details). The Bonferroni correction is overly conservative; it is not recommended. It is included to allow comparisons with the other methods.

Details

`Y` must contain quantitative data. They will be transformed to ranks within each column before computation of the coefficient of concordance.

The search for species associations described in Legendre (2005) proceeds in 3 steps:

(1) Correlation analysis of the species. A possible method is to compute Ward's agglomerative clustering of a matrix of correlations among the species. In detail: (1.1) compute a Pearson or Spearman correlation matrix (`correl.matrix`) among the species; (1.2) turn it into a distance matrix: `mat.D = as.dist(1-correl.matrix)`; (1.3) carry out Ward's hierarchical clustering of that matrix using `hclust`: `clust.ward = hclust(mat.D, "ward")`; (1.4) plot the dendrogram: `plot(clust.ward, hang=-1)`; (1.5) cut the dendrogram in two groups, retrieve the vector of species membership: `group.2 = cutree(clust.ward, k=2)`. (1.6) After steps 2 and 3 below, you may have to come back and try divisions of the species into k = $3, 4, 5, \ldots$ groups.

(2) Compute global tests of significance of the 2 (or more) groups using the function `kendall.global` and the vector defining the groups. Groups that are not globally significant must be refined or abandoned.

(3) Compute a posteriori tests of the contribution of individual species to the concordance of their group using the function `kendall.post` and the vector defining the groups. If some species have negative values for "Spearman.mean", this means that these species clearly do not belong to the group, hence that group is too inclusive. Go back to (1.5) and cut the dendrogram more finely. The left and right groups can be cut separately, independently of the levels along the dendrogram; write your own vector of group membership if `cutree` does not produce the desired groups.

The corrections used for multiple testing are applied to the list of P-values (P); they take into account the number of tests (k) carried out simultaneously (number of groups in `kendall.global`, or number of species in `kendall.post`). The corrections are performed using function `p.adjust`; see that function for the description of the correction methods. In addition, there is Šidák correction which defined as $P_{corr} = 1 - (1 - P)^k$.

Value

A table containing the following information in rows. The columns correspond to the groups of "judges" defined in vector "group". When function `Kendall.post` is used, there are as many tables as the number of predefined groups.

`W` Kendall's coefficient of concordance, W.

`F` F statistic. F = W*(m-1)/(1-W) where m is the number of judges.

`Prob.F` Probability associated with the F statistic, computed from the F distribution with nu1 = n-1-(2/m) and nu2 = nu1*(m-1); n is the number of objects.

`Corrected prob.F` Probabilities associated with F, corrected using the method selected in parameter `mult`. Shown only if there are more than one group.

`Chi2` Friedman's chi-square statistic (Friedman 1937) used in the permutation test of W.

`Prob.perm` Permutational probabilities, uncorrected.

`Corrected prob.perm` Permutational probabilities corrected using the method selected in parameter `mult`. Shown only if there are more than one group.

`Spearman.mean` Mean of the Spearman correlations between the judge under test and all the other judges in the same group.

`W.per.species` Contribution of the judge under test to the overall concordance statistic for that group.

Author(s)

F. Guillaume Blanchet, University of Alberta, and Pierre Legendre, Université de Montréal

References

Friedman, M. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32: 675-701.

Kendall, M. G. and B. Babington Smith. 1939. The problem of m rankings. Annals of Mathematical Statistics 10: 275-287.

Legendre, P. 2005. Species associations: the Kendall coefficient of concordance revisited. Journal of Agricultural, Biological, and Environmental Statistics 10: 226-245.

Legendre, P. 2009. Coefficient of concordance. In: Encyclopedia of Research Design. SAGE Publications (in press).

Siegel, S. and N. J. Castellan, Jr. 1988. Nonparametric statistics for the behavioral sciences. 2nd edition. McGraw-Hill, New York.

See Also

`cor`, `friedman.test`, `hclust`, `cutree`, `kmeans`, `cascadeKM`.

Examples

```
data(mite)
mite.hel <- decostand(mite, "hel")

# Reproduce the results shown in Table 2 of Legendre (2005), a single group
mite.small <- mite.hel[c(4,9,14,22,31,34,45,53,61,69),c(13:15,23)]
kendall.global(mite.small, nperm=49)
kendall.post(mite.small, mult="holm", nperm=49)

# Reproduce the results shown in Tables 3 and 4 of Legendre (2005), 2 groups
group <-c(1,1,2,1,1,1,1,1,2,1,1,1,1,1,1,2,1,2,1,1,1,1,2,1,2,1,1,1,1,1,2,2,2,2,2)
kendall.global(mite.hel, group=group, nperm=49)
kendall.post(mite.hel, group=group, mult="holm", nperm=49)

# NOTE: 'nperm' argument usually needs to be larger than 49.
# It was set to this low value for demonstration purposes.
```

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

Description

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

Usage

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

Arguments

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

Value

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

Note

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

Author(s)

Jari Oksanen with modifications by Gavin L. Simpson

Examples

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

## Expressions as labels
N <- 10     # Number of sites
df <- data.frame(Ca = rlnorm(N, 2), NO3 = rlnorm(N, 4),
                 SO4 = rlnorm(N, 10), K = rlnorm(N, 3))
ord <- rda(df, scale = TRUE)
### vector of expressions for labels
labs <- expression(Ca^{2+phantom()},
                   NO[3]^{-phantom()},
                   SO[4]^{2-phantom()},
                   K^{+phantom()})
scl <- "sites"
linestack(scores(ord, choices = 1, display = "species", scaling = scl),
          labels = labs, air = 2)
```

```
linestack(scores(ord, choices = 1, display = "site", scaling = scl),
          side = "left", add = TRUE)
title(main = "PCA axis 1")
```

---

make.cepnames *Abbreviates a Two-Part Botanical or Zoological Latin Name into Character String*

Description

Function is based on abbreviate, and will take given number of characters from the first (genus) and last (epithet) component of botanical or zoological Latin name and combine these into one shorter character string. The names will be unique and more characters will be used if needed. The default usage makes names with 4+4 characters popularized in Cornell Ecology Programs (CEP) and often known as CEP names. Abbreviated names are useful in ordination plots and other graphics to reduce clutter.

Usage

```
make.cepnames(names, minlengths = c(4,4), seconditem = FALSE,
   uniqgenera = FALSE, named = FALSE, method)
```

Arguments

names	The names to be abbreviated into a vector abbreviated names.
minlengths	The minimum lengths of first and second part of the abbreviation. If abbreviations are not unique, the parts can be longer.
seconditem	Take always the second part of the original name to the abbreviated name instead of the last part.
uniqgenera	Should the first part of the abbreviation (genus) also be unique. Unique genus can take space from the second part (epithet).
method	The abbreviate argument in last attempt to abbreviate the abbreviation. The default method tries to drop character from the end, but "both.sides" can remove characters from any position, including the genus part, and same genus can be abbreviated differently.
named	Should the result vector be named by original 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 current function produces 4+4 character CEP names as default, but it can also use other lengths. The function is based on abbreviate and will produce longer names if constructed names are not unique. If generic name is shorter than specified minimum length, more characters can be used by the epithet. If uniqgenera = TRUE, unique generic

name may need more characters than the minimum length, and these reduce the number of characters available for the epithet. The function drops characters from the end, but with `method = "both.sides"` the function tries to drop characters from other positions in the final attempt to abbreviate abbreviations, starting with lower-case vowels. The constructed names are based on binomial names with two components. If there is only one component (typically a genus), it can use also the space reserved for the second component. If there are more than two components of input names, the function selects only two: always the first, and either the last or second depending on the argument `seconditem`. Function will issue a warning of duplicated input names or of duplicated inferred binomial names, but resulting CEP names are made unique by adding a sequence number (with a warning listing these names).

Value

Function returns a vector of abbreviated names.

Note

The function does not handle Author names except strictly two-part names with `seconditem = TRUE`. It is often necessary to edit abbreviations manually.

Author(s)

Jari Oksanen

See Also

`abbreviate`.

Examples

```
names <- c("Aa maderoi", "Capsella bursa-pastoris", "Taraxacum",
  "Cladina rangiferina", "Cladonia rangiformis", "Cladonia cornuta",
  "Cladonia cornuta var. groenlandica", "Euphorbia cf. enopla",
  "Rumex acetosa", "Rumex acetosella")
make.cepnames(names)
make.cepnames(names, uniqgenera = TRUE)
make.cepnames(names, method = "both.sides")
## seconditem=TRUE makes subspecies duplicates
data(BCI)
names <- make.cepnames(names(BCI), seconditem=TRUE)
```

`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 matrices. The significance of the statistic is evaluated by permuting rows and columns of the first dissimilarity matrix. Test is one-sided and only tests that distances are positively correlated.

Usage

```
mantel(xdis, ydis, method="pearson", permutations=999, strata = NULL,
    na.rm = FALSE, parallel = getOption("mc.cores"))
mantel.partial(xdis, ydis, zdis, method = "pearson", permutations = 999,
    strata = NULL, na.rm = FALSE, parallel = getOption("mc.cores"))
## S3 method for class 'mantel'
summary(object, ...)
```

Arguments

`xdis`, `ydis`, `zdis`	Distance object of class `"dist"` or symmetric square matrices of distances. Only the lower triangle of square matrices is used. The first object `xdis` will be permuted in permutation tests.
`method`	Correlation method, as accepted by `cor`: `"pearson"`, `"spearman"` or `"kendall"`.
`permutations`	a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
`strata`	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
`na.rm`	Remove missing values in calculation of Mantel correlation. Use this option with care: Permutation tests can be biased, in particular if two matrices had missing values in matching positions.
`parallel`	Number of parallel processes or a predefined socket cluster. With `parallel = 1` uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
`object`	Result object.
`...`	Arguments passed to `summary.permustats` These include `alternative` to select the sidedness of the test.

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. Only the first matrix (`xdist`) will be permuted, and the second is kept constant. See `permutations` for additional details on permutation tests in Vegan.

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.

Borcard & Legendre (2012) warn against using partial Mantel test and recommend instead Mantel correlogram (mantel.correlog).

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. The distribution of permuted values can be inspected with permustats function.
permutations	Number of permutations.
control	A list of control values for the permutations as returned by the function how.

Author(s)

Jari Oksanen

References

Borcard, D. & Legendre, P. (2012) Is the Mantel correlogram powerful enough to be useful in ecological analysis? A simulation study. *Ecology* 93: 1473-1481.

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English Edition. Elsevier.

See Also

mantel.correlog.

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")
```

mantel.correlog *Mantel Correlogram*

Description

Function mantel.correlog computes a multivariate Mantel correlogram. Proposed by Sokal (1986) and Oden and Sokal (1986), the method is also described in Legendre and Legendre (2012, pp. 819–821) and tested and compared in Borcard and Legendere (2012).

Usage

```
mantel.correlog(D.eco, D.geo=NULL, XY=NULL, n.class=0, break.pts=NULL,
cutoff=TRUE, r.type="pearson", nperm=999, mult="holm", progressive=TRUE)
## S3 method for class 'mantel.correlog'
plot(x, alpha=0.05, ...)
```

Arguments

D.eco	An ecological distance matrix, with class either dist or matrix.
D.geo	A geographic distance matrix, with class either dist or matrix. Provide either D.geo or XY. Default: D.geo=NULL.
XY	A file of Cartesian geographic coordinates of the points. Default: XY=NULL.
n.class	Number of classes. If n.class=0, the Sturges equation will be used unless break points are provided.
break.pts	Vector containing the break points of the distance distribution. Provide (n.class+1) breakpoints, that is, a list with a beginning and an ending point. Default: break.pts=NULL.
cutoff	For the second half of the distance classes, cutoff = TRUE limits the correlogram to the distance classes that include all points. If cutoff = FALSE, the correlogram includes all distance classes.
r.type	Type of correlation in calculation of the Mantel statistic. Default: r.type="pearson". Other choices are r.type="spearman" and r.type="kendall", as in functions cor and mantel.
nperm	Number of permutations for the tests of significance. Default: nperm=999. For large data files, permutation tests are rather slow.
mult	Correct P-values for multiple testing. The correction methods are "holm" (default), "hochberg", "sidak", and other methods available in the p.adjust function: "bonferroni" (best known, but not recommended because it is overly conservative), "hommel", "BH", "BY", "fdr", and "none".
progressive	Default: progressive=TRUE for progressive correction of multiple-testing, as described in Legendre and Legendre (1998, p. 721). Test of the first distance class: no correction; second distance class: correct for 2 simultaneous tests; distance class k: correct for k simultaneous tests. progressive=FALSE: correct all tests for n.class simultaneous tests.
x	Output of mantel.correlog.

alpha	Significance level for the points drawn with black symbols in the correlogram. Default: alpha=0.05.
...	Other parameters passed from other functions.

Details

A correlogram is a graph in which spatial correlation values are plotted, on the ordinate, as a function of the geographic distance classes among the study sites along the abscissa. In a Mantel correlogram, a Mantel correlation (Mantel 1967) is computed between a multivariate (e.g. multi-species) distance matrix of the user's choice and a design matrix representing each of the geographic distance classes in turn. The Mantel statistic is tested through a permutational Mantel test performed by vegan's mantel function.

Borcard and Legendre (2012) show that the testing method in the Mantel correlogram has correct type I error and power, contrary to the simple and partial Mantel tests so often used by ecologists and geneticists in spatial analysis (see mantel.partial). They also show that the test in Mantel correlograms is the same test as used by Wagner (2004) in multiscale ordination (mso), and that it is closely related to the Geary's c test in univariate correlograms.

When a correction for multiple testing is applied, more permutations are necessary than in the no-correction case, to obtain significant p-values in the higher correlogram classes.

The print.mantel.correlog function prints out the correlogram. See examples.

Value

mantel.res	A table with the distance classes as rows and the class indices, number of distances per class, Mantel statistics (computed using Pearson's r, Spearman's r, or Kendall's tau), and p-values as columns. A positive Mantel statistic indicates positive spatial correlation. An additional column with p-values corrected for multiple testing is added unless mult="none".
n.class	The n umber of distance classes.
break.pts	The break points provided by the user or computed by the program.
mult	The name of the correction for multiple testing. No correction: mult="none".
progressive	A logical (TRUE, FALSE) value indicating whether or not a progressive correction for multiple testing was requested.
n.tests	The number of distance classes for which Mantel tests have been computed and tested for significance.
call	The function call.

Author(s)

Pierre Legendre, Université de Montréal

References

Borcard, D. & P. Legendre. 2012. Is the Mantel correlogram powerful enough to be useful in ecological analysis? A simulation study. Ecology 93: 1473-1481.

Legendre, P. and L. Legendre. 2012. Numerical ecology, 3rd English edition. Elsevier Science BV, Amsterdam.

Mantel, N. 1967. The detection of disease clustering and a generalized regression approach. Cancer Res. 27: 209-220.

Oden, N. L. and R. R. Sokal. 1986. Directional autocorrelation: an extension of spatial correlograms to two dimensions. Syst. Zool. 35: 608-617.

Sokal, R. R. 1986. Spatial data analysis and historical processes. 29-43 in: E. Diday et al. [eds.] Data analysis and informatics, IV. North-Holland, Amsterdam.

Sturges, H. A. 1926. The choice of a class interval. Journal of the American Statistical Association 21: 65–66.

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

Examples

```
# Mite data available in "vegan"
data(mite)
data(mite.xy)
mite.hel <- decostand(mite, "hellinger")

# Detrend the species data by regression on the site coordinates
mite.hel.resid <- resid(lm(as.matrix(mite.hel) ~ ., data=mite.xy))

# Compute the detrended species distance matrix
mite.hel.D <- dist(mite.hel.resid)

# Compute Mantel correlogram with cutoff, Pearson statistic
mite.correlog <- mantel.correlog(mite.hel.D, XY=mite.xy, nperm=49)
summary(mite.correlog)
mite.correlog
# or: print(mite.correlog)
# or: print.mantel.correlog(mite.correlog)
plot(mite.correlog)

# Compute Mantel correlogram without cutoff, Spearman statistic
mite.correlog2 <- mantel.correlog(mite.hel.D, XY=mite.xy, cutoff=FALSE,
   r.type="spearman", nperm=49)
summary(mite.correlog2)
mite.correlog2
plot(mite.correlog2)

# NOTE: 'nperm' argument usually needs to be larger than 49.
# It was set to this low value for demonstration purposes.
```

MDSaddpoints *Add New Points to NMDS ordination*

Description

Add new points to an existing metaMDS or monoMDS ordination.

Usage

```
MDSaddpoints(nmds, dis, neighbours = 5, maxit = 200)

dist2xy(dist, pick, type = c("xy", "xx"), invert = FALSE)
```

Arguments

nmds	Result object from metaMDS or monoMDS. The configuration of points is fixed, but new points are added.
dis	Rectangular non-symmetric dissimilarity matrix among new points (rows) and old fixed points (columns). Such matrix can be extracted from complete dissimilarities of both old and new points with dist2xy, or calculated with designdist2.
neighbours	Number of nearest points used to get the starting locations for new points.
maxit	Maximum number of iterations.
dist	Input dissimilarities.
pick	Indices (integers) of selected observations or a logical vector that is TRUE for picked items. The output will be in the original order and will not be reordered by this argument.
type	"xy" returns rectangular data of picked against not picked observations, and "xx" a subset of symmetric dissimilarities.
invert	Invert pick: drop elements listed.

Details

Function provides an interface to monoMDS Fortran code to add new points to an existing ordination that will be regarded as fixed. The function has a similar role as predict functions with newdata in Euclidean ordination (e.g. predict.cca). Input data must be a rectangular matrix of distances among new added points (rows) and all fixed old points (columns). Such matrices can be extracted from complete dissimilarities with helper function dist2xy. Function designdist2 can directly calculate such rectangular dissimilarity matrices between sampling units (rows) in two matries. In addition, **analogue** has distance function that can calculate dissimilarities among two matrices, including functions that cannot be specified in designdist2.

Great care is needed in preparing the dissimilarities for the input. The dissimilarity index must be exactly the same as in the fixed ordination, and columns must match old fixed points, and rows added new points.

Value

Function return a list of class "nmds" (there are no other objects of that type in **vegan**) with following elements

points	Coordinates of added new points
seeds	Starting coordinates for new points.
deltastress	Change of stress with added points.
iters	Number of iterations.
cause	Cause of termination of iterations. Integer for convergence criteria in monoMDS.

Examples

```
## Cross-validation: remove a point when performing NMDS and add as
## a new points
data(dune)
d <- vegdist(dune)
## remove point 3 from ordination
mod3 <- metaMDS(dist2xy(d, 3, "xx", invert = TRUE), trace=0)
## add point 3 to the result
MDSaddpoints(mod3, dist2xy(d, 3))
## Use designdist2
d15 <- designdist(dune[1:15,])
m15 <- metaMDS(d15, trace=0)
MDSaddpoints(m15, designdist2(dune[1:15,], dune[16:20,]))
```

MDSrotate *Rotate First MDS Dimension Parallel to an External Variable*

Description

Function rotates a multidimensional scaling result so that its first dimension is parallel to an external (environmental variable). The function can handle the results from metaMDS or monoMDS functions.

Usage

```
MDSrotate(object, vec, na.rm = FALSE, ...)
```

Arguments

object	A result object from metaMDS or monoMDS.
vec	An environmental variable or a matrix of such variables. The number of variables must be lower than the number of dimensions, and the solution is rotated to these variables in the order they appear in the matrix. Alternatively vec can be a factor, and the solution is rotated to optimal separation of factor levels using lda.
na.rm	Remove missing values from the continuous variable vec.
...	Other arguments (ignored).

Details

The orientation and rotation are undefined in multidimensional scaling. Functions metaMDS and metaMDS can rotate their solutions to principal components so that the dispersion of the points is highest on the first dimension. Sometimes a different rotation is more intuitive, and MDSrotate allows rotation of the result so that the first axis is parallel to a given external variable or two first variables are completely in a two-dimensional plane etc. If several external variables are supplied, they are applied in the order they are in the matrix. First axis is rotated to the first supplied variable, and the second axis to the second variable. Because variables are usually correlated, the second variable is not usually aligned with the second axis, but it is uncorrelated to later dimensions. There must be at least one free dimension: the number of external variables must be lower than the number of dimensions, and all used environmental variables are uncorrelated with that free dimension.

Alternatively the method can rotate to discriminate the levels of a factor using linear discriminant analysis (lda). This is hardly meaningful for two-dimensional solutions, since all rotations in two dimensions have the same separation of cluster levels. However, the function can be useful in finding a two-dimensional projection of clusters from more than two dimensions. The last dimension will always show the residual variation, and for k dimensions, only $k-1$ discrimination vectors are used.

Value

Function returns the original ordination result, but with rotated scores (both site and species if available), and the pc attribute of scores set to FALSE.

Note

Rotation to a factor variable is an experimental feature and may be removed. The discriminant analysis weights dimensions by their discriminating power, but MDSrotate performs a rigid rotation. Therefore the solution may not be optimal.

Author(s)

Jari Oksanen

See Also

metaMDS, monoMDS.

Examples

```
data(varespec)
data(varechem)
mod <- monoMDS(vegdist(varespec))
mod <- with(varechem, MDSrotate(mod, pH))
plot(mod)
ef <- envfit(mod ~ pH, varechem, permutations = 0)
plot(ef)
ordisurf(mod ~ pH, varechem, knots = 1, add = TRUE)
```

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

Description

Function metaMDS performs Nonmetric Multidimensional Scaling (NMDS), and tries to find a stable solution using several random starts. In addition, it standardizes the scaling in the result, so that the configurations are easier to interpret, and adds species scores to the site ordination. The metaMDS function does not provide actual NMDS, but it calls another function for the purpose. Currently monoMDS is the default choice, but it is also possible to call other functions as an engine.

Usage

```
metaMDS(comm, distance = "bray", k = 2, try = 20, trymax = 20,
    engine = monoMDS, autotransform =TRUE, noshare = FALSE, wascores = TRUE,
    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, cex = 0.7, ...)
## S3 method for class 'metaMDS'
points(x, display = c("sites", "species"),
    choices = c(1,2), shrink = FALSE, select, cex = 0.7, ...)
## S3 method for class 'metaMDS'
text(x, display = c("sites", "species"), labels,
    choices = c(1,2), shrink = FALSE, select, cex = 0.7, ...)
## S3 method for class 'metaMDS'
scores(x, display = c("sites", "species"), shrink = FALSE,
    choices, tidy = FALSE, ...)
metaMDSdist(comm, distance = "bray", autotransform = TRUE,
    noshare = TRUE, trace = 1, commname, zerodist = "ignore",
    distfun = vegdist, ...)
metaMDSiter(dist, k = 2, try = 20, trymax = 20, trace = 1, plot = FALSE,
    previous.best, engine = monoMDS, maker, parallel = getOption("mc.cores"),
    ...)
initMDS(x, k=2)
postMDS(X, dist, pc=TRUE, center=TRUE, halfchange, threshold=0.8,
    nthreshold=10, plot=FALSE, ...)
metaMDSredist(object, ...)
```

Arguments

comm	Community data. Alternatively, dissimilarities either as a dist structure or as a symmetric square matrix. In the latter case all other stages are skipped except random starts and centring and pc rotation of axes.
distance	Dissimilarity index used in vegdist.

k	Number of dimensions. NB., the number of points n should be $n > 2k + 1$, and preferably much higher in global non-metric MDS, and still higher in local NMDS.
try, trymax	Minimum and maximum numbers of random starts in search of stable solution. After try has been reached, the iteration will stop when similar solutions were repeated or trymax was reached.
engine	The function used for MDS. The default is to use the monoMDS function in **vegan**. It is also possible to use any MDS function which takes as three first arguments (in this order) input dissimilarities, matrix of initial configuration and number of dimensions, and returns a list with items stress and points for final configuration. See Examples for wrapping a compatible function.
autotransform	Use simple heuristics for possible data transformation of typical community data (see below). If you do not have community data, you should probably set autotransform = FALSE.
noshare	Triggering of calculation step-across or extended dissimilarities with function stepacross. The argument can be logical or a numerical value greater than zero and less than one. If TRUE, extended dissimilarities are used always when there are no shared species between some sites, if FALSE, they are never used. If noshare is a numerical value, stepacross is used when the proportion of site pairs with no shared species exceeds noshare. The number of pairs with no shared species is found with no.shared function, and noshare has no effect if input data were dissimilarities instead of community data.
wascores	Calculate species scores using function wascores.
expand	Expand weighted averages of species in wascores.
trace	Trace the function; trace = 2 or higher will be more voluminous.
plot	Graphical tracing: plot interim results. You may want to set par(ask = TRUE) with this option.
previous.best	Start searches from a previous solution. This can also be a monoMDS solution or a matrix of coordinates.
x	metaMDS result (or a dissimilarity structure for initMDS).
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.
cex	Character expansion for plotting symbols.
tidy	Return scores that are compatible with **ggvegan**: all scores are in a single data.frame, score type is identified by factor variable code ("sites" or "species", if available), the names by variable label. These scores are incompatible with conventional plot functions, but they can be used in **ggvegan** and **ggplot2**.
labels	Optional test to be used instead of row names. If select is used, labels are given only to selected items in the order they occur in the scores.
select	Items to be displayed. This can either be a logical vector which is TRUE for displayed items or a vector of indices of displayed items.

X	Configuration from multidimensional scaling.
commname	The name of comm: should not be given if the function is called directly.
zerodist	Handling of zero dissimilarities: either "fail" or "add" a small positive value, or "ignore". monoMDS and many other functions accept zero dissimilarities and the default is zerodist = "ignore", but with isoMDS you may need to set zerodist = "add".
distfun	Dissimilarity function. Any function returning a dist object and accepting argument method can be used (but some extra arguments may cause name conflicts).
maker	The name (character) of the engine. Only "monoMDS" has an effect and triggers some actions that are not known to be available with other engines.
parallel	Number of parallel processes or a predefined socket cluster. If you use predefined socket clusters (say, clus), you must issue clusterEvalQ(clus, library(vegan)) to make available internal **vegan** functions. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
dist	Dissimilarity matrix used in multidimensional scaling.
pc	Rotate to principal axes.
center	Centre the configuration.
halfchange	Scale axes to half-change units. This defaults TRUE when dissimilarities are known to have a theoretical maximum value (ceiling). Function vegdist will have that information in attribute maxdist, and for other distfun this is interpreted in a simple test (that can fail), and the information may not available when input data are distances. If FALSE, the ordination dissimilarities are scaled to the same range as the input dissimilarities.
threshold	Largest dissimilarity used in half-change scaling. If dissimilarities have a known (or inferred) ceiling, threshold is relative to that ceiling (see halfchange).
nthreshold	Minimum number of points in half-change scaling.
object	A result object from metaMDS.
...	Other parameters passed to functions. Function metaMDS passes all arguments to its component functions metaMDSdist, metaMDSiter, postMDS, and to distfun and engine.

Details

Non-metric Multidimensional Scaling (NMDS) is commonly regarded as the most robust unconstrained ordination method in community ecology (Minchin 1987). Function metaMDS is a wrapper function that calls several other functions to combine Minchin's (1987) recommendations into one command. The complete steps in metaMDS are:

1. Transformation: If the data values are larger than common abundance class scales, the function performs a Wisconsin double standardization (wisconsin). If the values look very large, the function also performs sqrt transformation. Both of these standardizations 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 standardize and transform data independently. The autotransform is intended for community data, and for other data types,

you should set autotransform = FALSE. This step is performed using metaMDSdist, and the step is skipped if input were dissimilarities.

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-Curtis 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. The default choice may be bad if you analyse other than community data, and you should probably select an appropriate index using argument distance. This step is performed using metaMDSdist, and the step is skipped if input were dissimilarities.
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 default NMDS engine is monoMDS which is able to break tied values at the maximum dissimilarity, and this is usually sufficient to handle cases with no shared species. stepacross is triggered by option noshare. If you do not like manipulation of original distances, you should set noshare = FALSE. This step is performed using metaMDSdist, and the step is skipped always when input were dissimilarities.
4. NMDS with random starts: NMDS easily gets trapped into local optima, and you must start NMDS several times from random starts to be confident that you have found the global solution. The strategy in metaMDS is to first run NMDS starting with the metric scaling (cmdscale which usually finds a good solution but often close to a local optimum), or use the previous.best solution if supplied, and take its solution as the standard (Run 0). Then metaMDS starts NMDS from several random starts (minimum number is given by try and maximum number by trymax). These random starts are generated by initMDS. 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 (function procrustes with option symmetric = TRUE). If the solutions are very similar in their Procrustes rmse and the largest residual is very small, the solutions are regarded as repeated and the better one is taken as the new standard. The conditions are stringent, and you may have found good and relatively similar 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. This step is performed using metaMDSiter. This is the first step performed if input data (comm) were dissimilarities. Random starts can be run with parallel processing (argument parallel).
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 (with function MDSrotate you can alternatively rotate the configuration so that the first axis is parallel to an environmental variable); 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 set by argument threshold). If there are enough points below this threshold (controlled by 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. If half-change scaling is not used, the ordination is scaled to the same range as the original dissimilarities. Half-change scaling is skipped by default if input

were dissimilarities, but can be turned on with argument `halfchange = TRUE`. NB., The PC rotation only changes the directions of reference axes, and it does not influence the configuration or solution in general.

6. Species scores: Function adds the species scores to the final solution as weighted averages using function `wascores` with given value of parameter `expand`. The expansion of weighted averages can be undone with `shrink = TRUE` in `plot` or `scores` functions, and the calculation of species scores can be suppressed with `wascores = FALSE`. This step is skipped if input were dissimilarities and community data were unavailable. However, the species scores can be added or replaced with `sppscores`.

Value

Function `metaMDS` returns an object of class `metaMDS` which inherits from the class of `engine`. The final site ordination is stored in the item `points`, and species ordination in the item `species`, and the stress in item `stress` (NB, the scaling of the stress depends on the `engine`: `isoMDS` uses percents, `monoMDS` and most other functions use proportions $0 \ldots 1$). The other items store the information on the steps taken and the items returned by the `engine` function. The object has `print`, `plot`, `points` and `text` methods. Functions `metaMDSdist` and `metaMDSredist` return `vegdist` objects. Function `initMDS` returns a random configuration. Functions `metaMDSiter` and `postMDS` returns the result of NMDS with updated configuration.

Results Could Not Be Repeated

Non-linear optimization is a hard task, and the best possible solution ("global optimum") may not be found from a random starting configuration. Most software solve this by starting from the result of metric scaling (`cmdscale`). This will probably give a good result, but not necessarily the "global optimum". **Vegan** does the same, but `metaMDS` tries to verify or improve this first solution ("try 0") using several random starts and seeing if the result can be repeated or improved and the improved solution repeated. If this does not succeed, you get a message that the result could not be repeated. However, the result will be at least as good as the usual standard strategy of starting from metric scaling or it may be improved. You may not need to do anything after such a message, but you can be satisfied with the result. If you want to be sure that you probably have a "global optimum" you may try the following instructions.

With default `engine = "monoMDS"` the function will tabulate the stopping criteria used, so that you can see which criterion should be made more stringent. The criteria can be given as arguments to `metaMDS` and their current values are described in `monoMDS`. In particular, if you reach the maximum number of iterations, you should increase the value of `maxit`. You may ask for a larger number of random starts without losing the old ones giving the previous solution in argument `previous.best`.

In addition to slack convergence criteria and too low number of random starts, wrong number of dimensions (argument `k`) is the most common reason for not being able to repeat similar solutions. NMDS is usually run with a low number dimensions (`k=2` or `k=3`), and for complex data increasing `k` by one may help. If you run NMDS with much higher number of dimensions (say, `k=10` or more), you should reconsider what you are doing and drastically reduce `k`. For very heterogeneous data sets with partial disjunctions, it may help to set `stepacross`, but for most data sets the default `weakties = TRUE` is sufficient.

Please note that you can give all arguments of other `metaMDS*` functions and NMDS engine (default `monoMDS`) in your `metaMDS` command,and you should check documentation of these functions for details.

Common Wrong Claims

NMDS is often misunderstood and wrong claims of its properties are common on the Web and even in publications. It is often claimed that the NMDS configuration is non-metric which means that you cannot fit environmental variables or species onto that space. This is a false statement. In fact, the result configuration of NMDS is metric, and it can be used like any other ordination result. In NMDS the rank orders of Euclidean distances among points in ordination have a non-metric monotone relationship to any observed dissimilarities. The transfer function from observed dissimilarities to ordination distances is non-metric (Kruskal 1964a, 1964b), but the ordination result configuration is metric and observed dissimilarities can be of any kind (metric or non-metric).

The ordination configuration is usually rotated to principal axes in `metaMDS`. The rotation is performed after finding the result, and it only changes the direction of the reference axes. Before rotation the directions of axes are arbitrary, and the same solution (same configuration, same stress) the orientation of axes is arbitrary. The only important feature in the NMDS solution are the ordination distances, and these do not change in rotation. Similarly, the rank order of distances does not change in uniform scaling or centring of configuration of points. You can also rotate the NMDS solution to external environmental variables with `MDSrotate`. This rotation will also only change the orientation of axes, but will not change the configuration of points or distances between points in ordination space.

Function `stressplot` displays the method graphically: it plots the observed dissimilarities against distances in ordination space, and also shows the non-metric monotone regression.

Note

Function `metaMDS` is a simple wrapper for an NMDS engine (either `monoMDS` or any compatible function, and some support functions (`metaMDSdist`, `stepacross`, `metaMDSiter`, `initMDS`, `postMDS`, `wascores`). You can call these support functions separately for the full control of results. Data transformation, dissimilarities and possible `stepacross` are made in function `metaMDSdist` which returns a dissimilarity result. Iterative search (with starting values from `initMDS` with selected `engine` is made in `metaMDSiter`. Post-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 may not save the used dissimilarity matrix (`monoMDS` does), but `metaMDSredist` tries to reconstruct the used dissimilarities with original data transformation and possible `stepacross`.

The `metaMDS` function was designed to be used with community data. If you have other type of data, you should probably set some arguments to non-default values: probably at least `wascores`, `autotransform` and `noshare` should be `FALSE`. If you have negative data entries, `metaMDS` will set the previous to `FALSE` with a warning.

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.

Kruskal, J.B. (1964a). Multidimensional scaling by optimizing goodness-of-fit to a nonmetric hypothesis. *Psychometrika* 29, 1–28.

Kruskal, J.B. (1964b). Nonmetric multidimensional scaling: a numerical method. *Psychometrika* 29, 115–129.

Minchin, P.R. (1987). An evaluation of relative robustness of techniques for ecological ordinations. *Vegetatio* 69, 89–107.

See Also

monoMDS, decostand, wisconsin, vegdist, rankindex, stepacross, procrustes, wascores, sppscores, MDSrotate, ordiplot, stressplot.

Examples

```
## The recommended way of running NMDS (Minchin 1987)
##
data(dune)
## IGNORE_RDIFF_BEGIN
## Global NMDS using monoMDS
sol <- metaMDS(dune)
sol
plot(sol, type="t", optimize = TRUE)
## Start from previous best solution
sol <- metaMDS(dune, previous.best = sol)
## Local NMDS and stress 2 of monoMDS
sol2 <- metaMDS(dune, model = "local", stress=2)
sol2
## Use Arrhenius exponent 'z' as a binary dissimilarity measure
sol <- metaMDS(dune, distfun = betadiver, distance = "z")
sol
## IGNORE_RDIFF_END
## Wrap package smacof function mds as engine (you must load smacof first)
smacof <- function(dist, y, k, ...) {
   m <- mds(delta = dist, init = y, ndim = k, ...)
   m$points <- m$conf
   m
}
## use this as metaMDS(..., engine = smacof, type = "ordinal")
```

---

mite *Oribatid Mite Data with Explanatory Variables*

Description

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

Usage

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

Format

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

The fields in the environmental data are:

SubsDens Substrate density (g/L)

WatrCont Water content of the substrate (g/L)

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

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

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

Source

Pierre Legendre

References

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

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

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

Examples

```
data(mite)
```

monoMDS *Global and Local Non-metric Multidimensional Scaling and Linear and Hybrid Scaling*

Description

Function implements Kruskal's (1964a,b) non-metric multidimensional scaling (NMDS) using monotone regression and primary ("weak") treatment of ties. In addition to traditional global NMDS, the function implements local NMDS, linear and hybrid multidimensional scaling.

Usage

```
monoMDS(dist, y, k = 2, model = c("global", "local", "linear", "hybrid"),
    threshold = 0.8, maxit = 500, weakties = TRUE, stress = 1,
    scaling = TRUE, pc = TRUE, smin = 1e-4, sfgrmin = 1e-7,
    sratmax=0.999999, ...)
## S3 method for class 'monoMDS'
scores(x, display = "sites", shrink = FALSE, choices,
    tidy = FALSE, ...)
## S3 method for class 'monoMDS'
plot(x, display = "sites", choices = c(1,2), type = "t", ...)
## S3 method for class 'monoMDS'
points(x, display = "sites", choices = c(1,2), select, ...)
## S3 method for class 'monoMDS'
text(x, display = "sites", labels, choices = c(1,2),
    select, ...)
```

Arguments

`dist`	Input dissimilarities.
`y`	Starting configuration. A random configuration will be generated if this is missing.
`k`	Number of dimensions. NB., the number of points n should be $n > 2k + 1$, and preferably higher in non-metric MDS.
`model`	MDS model: `"global"` is normal non-metric MDS with a monotone regression, `"local"` is non-metric MDS with separate regressions for each point, `"linear"` uses linear regression, and `"hybrid"` uses linear regression for dissimilarities below a threshold in addition to monotone regression. See Details.
`threshold`	Dissimilarity below which linear regression is used alternately with monotone regression.
`maxit`	Maximum number of iterations.
`weakties`	Use primary or weak tie treatment, where equal observed dissimilarities are allowed to have different fitted values. if `FALSE`, then secondary (strong) tie treatment is used, and tied values are not broken.
`stress`	Use stress type 1 or 2 (see Details).

scaling	Scale final scores to unit root mean squares.
pc	Rotate final scores to principal components.
smin, sfgrmin, sratmax	Convergence criteria: iterations stop when stress drops below smin, scale factor of the gradient drops below sfgrmin, or stress ratio between two iterations goes over sratmax (but is still < 1).
x	A monoMDS result.
display	Kind of scores. Normally there are only scores for "sites", but "species" scores can be added with sppscores.
shrink	Shrink back species scores if they were expanded in wascores.
tidy	Return scores that are compatible with **ggvegan**: all scores are in a single data.frame, score type is identified by factor variable code ("sites" or "species", if available), the names by variable label. These scores are incompatible with conventional plot functions, but they can be used in **ggvegan** and **ggplot2**.
choices	Dimensions returned or plotted. The default NA returns all dimensions.
type	The type of the plot: "t" for text, "p" for points, and "n" for none.
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.
labels	Labels to be use used instead of row names. If select is used, labels are given only the selected items in the order they occur in the scores.
...	Other parameters to the functions (ignored in monoMDS, passed to graphical functions in plot.).

Details

There are several versions of non-metric multidimensional scaling in R, but monoMDS offers the following unique combination of features:

- "Weak" treatment of ties (Kruskal 1964a,b), where tied dissimilarities can be broken in monotone regression. This is especially important for cases where compared sites share no species and dissimilarities are tied to their maximum value of one. Breaking ties allows these points to be at different distances and can help in recovering very long coenoclines (gradients). Functions in the **smacof** package also have adequate tie treatment.
- Handles missing values in a meaningful way.
- Offers "local" and "hybrid" scaling in addition to usual "global" NMDS (see below).
- Uses fast compiled code (isoMDS of the **MASS** package also uses compiled code).

Function monoMDS uses Kruskal's (1964b) original monotone regression to minimize the stress. There are two alternatives of stress: Kruskal's (1964a,b) original or "stress 1" and an alternative version or "stress 2" (Sibson 1972). Both of these stresses can be expressed with a general formula

$$s^2 = \frac{\sum(d - \hat{d})^2}{\sum(d - d_0)^2}$$

where d are distances among points in ordination configuration, $\hat{d}$ are the fitted ordination distances, and d_0 are the ordination distances under null model. For "stress 1" $d_0 = 0$, and for "stress 2" $d_0 =$

$\bar{d}$ or mean distances. "Stress 2" can be expressed as $s^2 = 1 - R^2$, whereR^2 is squared correlation between fitted values and ordination distances, and so related to the "linear fit" of `stressplot`.

Function `monoMDS` can fit several alternative NMDS variants that can be selected with argument `model`. The default `model = "global"` fits global NMDS, or Kruskal's (1964a,b) original NMDS similar to `isoMDS` (**MASS**). Alternative `model = "local"` fits local NMDS where independent monotone regression is used for each point (Sibson 1972). Alternative `model = "linear"` fits a linear MDS. This fits a linear regression instead of monotone, and is not identical to metric scaling or principal coordinates analysis (`cmdscale`) that performs an eigenvector decomposition of dissimilarities (Gower 1966). Alternative `model = "hybrid"` implements hybrid MDS that uses monotone regression for all points and linear regression for dissimilarities below or at a `threshold` dissimilarity in alternating steps (Faith et al. 1987). Function `stressplot` can be used to display the kind of regression in each `model`.

Scaling, orientation and direction of the axes is arbitrary. However, the function always centres the axes, and the default `scaling` is to scale the configuration of unit root mean square and to rotate the axes (argument `pc`) to principal components so that the first dimension shows the major variation. It is possible to rotate the solution so that the first axis is parallel to a given environmental variable using function `MDSrotate`.

Value

Returns an object of class `"monoMDS"`. The final scores are returned in item `points` (function `scores` extracts these results), and the stress in item `stress`. In addition, there is a large number of other items (but these may change without notice in the future releases). There are no species scores, but these can be added with `sppscores` function.

Convergence Criteria

NMDS is iterative, and the function stops when any of its convergence criteria is met. There is actually no criterion of assured convergence, and any solution can be a local optimum. You should compare several random starts (or use `monoMDS` via `metaMDS`) to assess if the solutions is likely a global optimum.

The stopping criteria are:

`maxit`**:** Maximum number of iterations. Reaching this criterion means that solutions was almost certainly not found, and `maxit` should be increased.

`smin`**:** Minimum stress. If stress is nearly zero, the fit is almost perfect. Usually this means that data set is too small for the requested analysis, and there may be several different solutions that are almost as perfect. You should reduce the number of dimensions (`k`), get more data (more observations) or use some other method, such as metric scaling (`cmdscale`, `wcmdscale`).

`sratmax`**:** Change in stress. Values close to one mean almost unchanged stress. This may mean a solution, but it can also signal stranding on suboptimal solution with flat stress surface.

`sfgrmin`**:** Minimum scale factor. Values close to zero mean almost unchanged configuration. This may mean a solution, but will also happen in local optima.

Note

This is the default NMDS function used in `metaMDS`. Function `metaMDS` adds support functions so that NMDS can be run like recommended by Minchin (1987).

Author(s)

Peter R. Michin (Fortran core) and Jari Oksanen (R interface).

References

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

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

Kruskal, J.B. 1964a. Multidimensional scaling by optimizing goodness-of-fit to a nonmetric hypothesis. *Psychometrika* 29, 1–28.

Kruskal, J.B. 1964b. Nonmetric multidimensional scaling: a numerical method. *Psychometrika* 29, 115–129.

Minchin, P.R. 1987. An evaluation of relative robustness of techniques for ecological ordinations. *Vegetatio* 69, 89–107.

Sibson, R. 1972. Order invariant methods for data analysis. *Journal of the Royal Statistical Society B* 34, 311–349.

See Also

`metaMDS` for the **vegan** way of running NMDS, and `isoMDS` and **smacof** for some alternative implementations of NMDS.

Examples

```
data(dune)
dis <- vegdist(dune)
m <- monoMDS(dis, model = "loc")
m
plot(m)
```

MOStest — *Mitchell-Olds and Shaw Test for the Location of Quadratic Extreme*

Description

Mitchell-Olds & Shaw test concerns the location of the highest (hump) or lowest (pit) value of a quadratic curve at given points. Typically, it is used to study whether the quadratic hump or pit is located within a studied interval. The current test is generalized so that it applies generalized linear models (`glm`) with link function instead of simple quadratic curve. The test was popularized in ecology for the analysis of humped species richness patterns (Mittelbach et al. 2001), but it is more general. With logarithmic link function, the quadratic response defines the Gaussian response model of ecological gradients (ter Braak & Looman 1986), and the test can be used for inspecting the location of Gaussian optimum within a given range of the gradient. It can also be used to replace Tokeshi's test of "bimodal" species frequency distribution.

Usage

```
MOStest(x, y, interval, ...)
## S3 method for class 'MOStest'
plot(x, which = c(1,2,3,6), ...)
fieller.MOStest(object, level = 0.95)
## S3 method for class 'MOStest'
profile(fitted, alpha = 0.01, maxsteps = 10, del = zmax/5, ...)
## S3 method for class 'MOStest'
confint(object, parm = 1, level = 0.95, ...)
```

Arguments

`x`	The independent variable or plotting object in `plot`.
`y`	The dependent variable.
`interval`	The two points at which the test statistic is evaluated. If missing, the extremes of `x` are used.
`which`	Subset of plots produced. Values `which=1` and `2` define plots specific to `MOStest` (see Details), and larger values select graphs of `plot.lm` (minus 2).
`object`, `fitted`	A result object from `MOStest`.
`level`	The confidence level required.
`alpha`	Maximum significance level allowed.
`maxsteps`	Maximum number of steps in the profile.
`del`	A step length parameter for the profile (see code).
`parm`	Ignored.
`...`	Other variables passed to functions. Function `MOStest` passes these to `glm` so that these can include `family`. The other functions pass these to underlying graphical functions.

Details

The function fits a quadratic curve $\mu = b_0 + b_1 x + b_2 x^2$ with given `family` and link function. If $b_2 < 0$, this defines a unimodal curve with highest point at $u = -b_1/(2b_2)$ (ter Braak & Looman 1986). If $b_2 > 0$, the parabola has a minimum at u and the response is sometimes called "bimodal". The null hypothesis is that the extreme point u is located within the interval given by points p_1 and p_2. If the extreme point u is exactly at p_1, then $b_1 = 0$ on shifted axis $x - p_1$. In the test, origin of `x` is shifted to the values p_1 and p_2, and the test statistic is based on the differences of deviances between the original model and model where the origin is forced to the given location using the standard `anova.glm` function (Oksanen et al. 2001). Mitchell-Olds & Shaw (1987) used the first degree coefficient with its significance as estimated by the `summary.glm` function. This give identical results with Normal error, but for other error distributions it is preferable to use the test based on differences in deviances in fitted models.

The test is often presented as a general test for the location of the hump, but it really is dependent on the quadratic fitted curve. If the hump is of different form than quadratic, the test may be insignificant.

Because of strong assumptions in the test, you should use the support functions to inspect the fit. Function `plot(..., which=1)` displays the data points, fitted quadratic model, and its approximate 95% confidence intervals (2 times SE). Function `plot` with `which = 2` displays the approximate confidence interval of the polynomial coefficients, together with two lines indicating the combinations of the coefficients that produce the evaluated points of `x`. Moreover, the cross-hair shows the approximate confidence intervals for the polynomial coefficients ignoring their correlations. Higher values of `which` produce corresponding graphs from `plot.lm`. That is, you must add 2 to the value of `which` in `plot.lm`.

Function `fieller.MOStest` approximates the confidence limits of the location of the extreme point (hump or pit) using Fieller's theorem following ter Braak & Looman (1986). The test is based on quasideviance except if the `family` is `poisson` or `binomial`. Function `profile` evaluates the profile deviance of the fitted model, and `confint` finds the profile based confidence limits following Oksanen et al. (2001).

The test is typically used in assessing the significance of diversity hump against productivity gradient (Mittelbach et al. 2001). It also can be used for the location of the pit (deepest points) instead of the Tokeshi test. Further, it can be used to test the location of the the Gaussian optimum in ecological gradient analysis (ter Braak & Looman 1986, Oksanen et al. 2001).

Value

The function is based on `glm`, and it returns the result of object of `glm` amended with the result of the test. The new items in the `MOStest` are:

`isHump`	`TRUE` if the response is a hump.
`isBracketed`	`TRUE` if the hump or the pit is bracketed by the evaluated points.
`hump`	Sorted vector of location of the hump or the pit and the points where the test was evaluated.
`coefficients`	Table of test statistics and their significances.

Note

Function `fieller.MOStest` is based on package **optgrad** in the Ecological Archives doi:10.6084/m9.figshare.3521975 accompanying Oksanen et al. (2001). The Ecological Archive package **optgrad** also contains profile deviance method for the location of the hump or pit, but the current implementation of `profile` and `confint` rather follow the example of `profile.glm` and `confint.glm`.

Author(s)

Jari Oksanen

References

Mitchell-Olds, T. & Shaw, R.G. 1987. Regression analysis of natural selection: statistical inference and biological interpretation. *Evolution* 41, 1149–1161.

Mittelbach, G.C. Steiner, C.F., Scheiner, S.M., Gross, K.L., Reynolds, H.L., Waide, R.B., Willig, R.M., Dodson, S.I. & Gough, L. 2001. What is the observed relationship between species richness and productivity? *Ecology* 82, 2381–2396.

Oksanen, J., Läärä, E., Tolonen, K. & Warner, B.G. 2001. Confidence intervals for the optimum in the Gaussian response function. *Ecology* 82, 1191–1197.

ter Braak, C.J.F & Looman, C.W.N 1986. Weighted averaging, logistic regression and the Gaussian response model. *Vegetatio* 65, 3–11.

See Also

The no-interaction model can be fitted with `humpfit` in the github package **natto**.

Examples

```
## The Al-Mufti data analysed in humpfit():
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)
mod <- MOStest(mass, spno)
## Insignificant
mod
## ... but inadequate shape of the curve
op <- par(mfrow=c(2,2), mar=c(4,4,1,1)+.1)
plot(mod)
## Looks rather like log-link with Poisson error and logarithmic biomass
mod <- MOStest(log(mass), spno, family=quasipoisson)
mod
plot(mod)
par(op)
## Confidence Limits
fieller.MOStest(mod)
confint(mod)
plot(profile(mod))
```

`mrpp` *Multi Response Permutation Procedure and Mean Dissimilarity Matrix*

Description

Multiple Response Permutation Procedure (MRPP) provides a test of whether there is a significant difference between two or more groups of sampling units. Function `meandist` finds the mean within and between block dissimilarities.

Usage

```
mrpp(dat, grouping, permutations = 999, distance = "euclidean",
     weight.type = 1, strata = NULL, parallel = getOption("mc.cores"))
meandist(dist, grouping, ...)
## S3 method for class 'mrpp'
summary(object, ...)
## S3 method for class 'meandist'
summary(object, ...)
```

```
## S3 method for class 'meandist'
plot(x, kind = c("dendrogram", "histogram"),  cluster = "average",
     ylim, axes = TRUE, ...)
```

Arguments

dat	data matrix or data frame in which rows are samples and columns are response variable(s), or a dissimilarity object or a symmetric square matrix of dissimilarities.
grouping	Factor or numeric index for grouping observations.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices. These are used to assess the significance of the MRPP statistic, $delta$.
distance	Choice of distance metric that measures the dissimilarity between two observations . See vegdist for options. This will be used if dat was not a dissimilarity structure of a symmetric square matrix.
weight.type	choice of group weights. See Details below for options.
strata	An integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata.
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
dist	A dist object of dissimilarities, such as produced by functions dist, vegdist or designdist.
.	
object, x	A mrpp or meandist result object.
kind	Draw a dendrogram or a histogram; see Details.
cluster	A clustering method for the hclust function for kind = "dendrogram". Any hclust method can be used, but perhaps only "average" and "single" make sense.
ylim	Limits for vertical axes (optional).
axes	Draw scale for the vertical axis.
...	Further arguments passed to functions.

Details

Multiple Response Permutation Procedure (MRPP) provides a test of whether there is a significant difference between two or more groups of sampling units. This difference may be one of location (differences in mean) or one of spread (differences in within-group distance; cf. Warton et al. 2012). Function mrpp operates on a data.frame matrix where rows are observations and responses data matrix. The response(s) may be uni- or multivariate. The method is philosophically and mathematically allied with analysis of variance, in that it compares dissimilarities within and among groups. If two groups of sampling units are really different (e.g. in their species composition), then

average of the within-group compositional dissimilarities ought to be less than the average of the dissimilarities between two random collection of sampling units drawn from the entire population.

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

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

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

Function `meandist` calculates a matrix of mean within-cluster dissimilarities (diagonal) and between-cluster dissimilarities (off-diagonal elements), and an attribute `n` of `grouping` counts. Function `summary` finds the within-class, between-class and overall means of these dissimilarities, and the MRPP statistics with all `weight.type` options and the Classification Strength, CS (Van Sickle and Hughes, 2000). CS is defined for dissimilarities as $\bar{B} - \bar{W}$, where $\bar{B}$ is the mean between cluster dissimilarity and $\bar{W}$ is the mean within cluster dissimilarity with `weight.type = 1`. The function does not perform significance tests for these statistics, but you must use `mrpp` with appropriate `weight.type`. There is currently no significance test for CS, but `mrpp` with `weight.type = 1` gives the correct test for $\bar{W}$ and a good approximation for CS. Function `plot` draws a dendrogram or a histogram of the result matrix based on the within-group and between group dissimilarities. The dendrogram is found with the method given in the `cluster` argument using function `hclust`. The terminal segments hang to within-cluster dissimilarity. If some of the clusters are more heterogeneous than the combined class, the leaf segment are reversed. The histograms are based on dissimilarities, but ore otherwise similar to those of Van Sickle and Hughes (2000): horizontal line is drawn at the level of mean between-cluster dissimilarity and vertical lines connect within-cluster dissimilarities to this line.

Value

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

`call`	Function call.
`delta`	The overall weighted mean of group mean distances.
`E.delta`	expected delta, under the null hypothesis of no group structure. This is the mean of original dissimilarities.
`CS`	Classification strength (Van Sickle and Hughes, 2000). Currently not implemented and always `NA`.
`n`	Number of observations in each class.
`classdelta`	Mean dissimilarities within classes. The overall δ is the weighted average of these values with given `weight.type`

.

`Pvalue`	Significance of the test.
`A`	A chance-corrected estimate of the proportion of the distances explained by group identity; a value analogous to a coefficient of determination in a linear model.
`distance`	Choice of distance metric used; the "method" entry of the dist object.
`weight.type`	The choice of group weights used.
`boot.deltas`	The vector of "permuted deltas," the deltas calculated from each of the permuted datasets. The distribution of this item can be inspected with `permustats` function.
`permutations`	The number of permutations used.
`control`	A list of control values for the permutations as returned by the function `how`.

Note

This difference may be one of location (differences in mean) or one of spread (differences in within-group distance). That is, it may find a significant difference between two groups simply because one of those groups has a greater dissimilarities among its sampling units. Most `mrpp` models can be analysed with `adonis2` which seems not suffer from the same problems as `mrpp` and is a more robust alternative.

Author(s)

M. Henry H. Stevens <HStevens@muohio.edu> and Jari Oksanen.

References

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

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

J. Van Sickle and R. M. Hughes 2000. Classification strengths of ecoregions, catchments, and geographic clusters of aquatic vertebrates in Oregon. *J. N. Am. Benthol. Soc.* 19:370–384.

Warton, D.I., Wright, T.W., Wang, Y. 2012. Distance-based multivariate analyses confound location and dispersion effects. *Methods in Ecology and Evolution*, 3, 89–101

See Also

`anosim` for a similar test based on ranks, and `mantel` for comparing dissimilarities against continuous variables, and `vegdist` for obtaining dissimilarities, `adonis2` is a more robust alternative in most cases.

Examples

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

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

plot(dune.ord <- metaMDS(dune, trace=0), type="text", display="sites" )
with(dune.env, ordihull(dune.ord, Management))

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

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

Description

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

Usage

```
mso(object.cca, object.xy, grain = 1, round.up = FALSE, permutations = 0)
msoplot(x, alpha = 0.05, explained = FALSE, ylim = NULL, legend = "topleft", ...)
```

Arguments

`object.cca`	An object of class cca, created by the `cca` or `rda` function.

object.xy	A vector, matrix or data frame with the spatial coordinates of the data represented by object.cca. The number of rows must match the number of observations (as given by nobs) in cca.object. Alternatively, interpoint distances can be supplied as a dist object.
grain	Interval size for distance classes.
round.up	Determines the choice of breaks. If false, distances are rounded to the nearest multiple of grain. If true, distances are rounded to the upper multiple of grain.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
x	A result object of mso.
alpha	Significance level for the two-sided permutation test of the Mantel statistic for spatial independence of residual inertia and for the point-wise envelope of the variogram of the total variance. A Bonferroni-type correction can be achieved by dividing the overall significance value (e.g. 0.05) by the number of distance classes.
explained	If false, suppresses the plotting of the variogram of explained variance.
ylim	Limits for y-axis.
legend	The x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by legend.
...	Other arguments passed to functions.

Details

The Mantel test is an adaptation of the function mantel to the parallel testing of several distance classes and similar to multivariate mantel.correlog. It compares the mean inertia in each distance class to the pooled mean inertia of all other distance classes.

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

Value

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

grain	The grain attribute defines the interval size of the distance classes .
H	H is an object of class 'dist' and contains the geographic distances between observations.
H.test	H.test contains a set of dummy variables that describe which pairs of observations (rows = elements of object$H) fall in which distance class (columns).
vario	The vario attribute is a data frame that contains some or all of the following components for the rda case (cca case in brackets):
	H Distance class as multiples of grain.

`Dist` Average distance of pairs of observations in distance class H.

n Number of unique pairs of observations in distance class H.

`All` Empirical (chi-square) variogram of total variance (inertia).

`Sum` Sum of empirical (chi-square) variograms of explained and residual variance (inertia).

`CA` Empirical (chi-square) variogram of residual variance (inertia).

`CCA` Empirical (chi-square) variogram of explained variance (inertia).

`pCCA` Empirical (chi-square) variogram of conditioned variance (inertia).

`se` Standard error of the empirical (chi-square) variogram of total variance (inertia).

`CA.signif` P-value of permutation test for spatial independence of residual variance (inertia).

Note

The function is based on the code published in the Ecological Archives E085-006 (doi:10.1890/020738).

Author(s)

The responsible author was Helene Wagner.

References

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

See Also

`mantel.correlog`.

Examples

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

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

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

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

mite.cca <- cca(log(mite + 1))
mite.cca <- mso(mite.cca, mite.xy, grain =  1, permutations = 99)
msoplot(mite.cca)
mite.cca

## Constrained ordination with test for residual autocorrelation
## and scale-invariance of species-environment relationships
mite.cca <- cca(log(mite + 1) ~ SubsDens + WatrCont + Substrate + Shrub + Topo, mite.env)
mite.cca <- mso(mite.cca, mite.xy, permutations = 99)
msoplot(mite.cca)
mite.cca
```

`multipart` *Multiplicative Diversity Partitioning*

Description

In multiplicative diversity partitioning, mean values of alpha diversity at lower levels of a sampling hierarchy are compared to the total diversity in the entire data set or the pooled samples (gamma diversity).

Usage

```
multipart(...)
## Default S3 method:
multipart(y, x, index=c("renyi", "tsallis"), scales = 1,
    global = FALSE, relative = FALSE, nsimul=99, method = "r2dtable", ...)
## S3 method for class 'formula'
multipart(formula, data, index=c("renyi", "tsallis"), scales = 1,
    global = FALSE, relative = FALSE, nsimul=99, method = "r2dtable", ...)
```

Arguments

`y`	A community matrix.
`x`	A matrix with same number of rows as in `y`, columns coding the levels of sampling hierarchy. The number of groups within the hierarchy must decrease from left to right. If `x` is missing, two levels are assumed: each row is a group in the first level, and all rows are in the same group in the second level.
`formula`	A two sided model formula in the form `y ~ x`, where `y` is the community data matrix with samples as rows and species as column. Right hand side (`x`) must be grouping variable(s) referring to levels of sampling hierarchy, terms from right to left will be treated as nested (first column is the lowest, last is the highest level). The formula will add a unique identifier to rows and constant for the rows to always produce estimates of row-level alpha and overall gamma diversities. You must use non-formula interface to avoid this behaviour. Interaction terms are not allowed.

data	A data frame where to look for variables defined in the right hand side of formula. If missing, variables are looked in the global environment.
index	Character, the entropy index to be calculated (see Details).
relative	Logical, if TRUE then beta diversity is standardized by its maximum (see Details).
scales	Numeric, of length 1, the order of the generalized diversity index to be used.
global	Logical, indicates the calculation of beta diversity values, see Details.
nsimul	Number of permutations to use. If nsimul = 0, only the FUN argument is evaluated. It is thus possible to reuse the statistic values without a null model.
method	Null model method: either a name (character string) of a method defined in make.commsim or a commsim function. The default "r2dtable" keeps row sums and column sums fixed. See oecosimu for Details and Examples.
...	Other arguments passed to oecosimu, i.e. method, thin or burnin.

Details

Multiplicative diversity partitioning is based on Whittaker's (1972) ideas, that has recently been generalised to one parametric diversity families (i.e. Rényi and Tsallis) by Jost (2006, 2007). Jost recommends to use the numbers equivalents (Hill numbers), instead of pure diversities, and proofs, that this satisfies the multiplicative partitioning requirements.

The current implementation of multipart calculates Hill numbers based on the functions renyi and tsallis (provided as index argument). If values for more than one scales are desired, it should be done in separate runs, because it adds extra dimensionality to the implementation, which has not been resolved efficiently.

Alpha diversities are then the averages of these Hill numbers for each hierarchy levels, the global gamma diversity is the alpha value calculated for the highest hierarchy level. When global = TRUE, beta is calculated relative to the global gamma value:

$$\beta_i = \gamma/\alpha_i$$

when global = FALSE, beta is calculated relative to local gamma values (local gamma means the diversity calculated for a particular cluster based on the pooled abundance vector):

$$\beta_i j = \alpha_{(i+1)j}/mean(\alpha_{ij})$$

where j is a particular cluster at hierarchy level i. Then beta diversity value for level i is the mean of the beta values of the clusters at that level, $\beta_i = mean(\beta_{ij})$.

If relative = TRUE, the respective beta diversity values are standardized by their maximum possible values ($mean(\beta_{ij})/\beta_{max,ij}$) given as $\beta_{max,ij} = n_j$ (the number of lower level units in a given cluster j).

The expected diversity components are calculated nsimul times by individual based randomization of the community data matrix. This is done by the "r2dtable" method in oecosimu by default.

Value

An object of class "multipart" with same structure as "oecosimu" objects.

Author(s)

Péter Sólymos, <solymos@ualberta.ca>

References

Jost, L. (2006). Entropy and diversity. *Oikos*, **113**, 363–375.

Jost, L. (2007). Partitioning diversity into independent alpha and beta components. *Ecology*, **88**, 2427–2439.

Whittaker, R. (1972). Evolution and measurement of species diversity. *Taxon*, **21**, 213–251.

See Also

See adipart for additive diversity partitioning, hiersimu for hierarchical null model testing and oecosimu for permutation settings and calculating p-values.

Examples

```
## NOTE: 'nsimul' argument usually needs to be >= 99
## here much lower value is used for demonstration

data(mite)
data(mite.xy)
data(mite.env)
## Function to get equal area partitions of the mite data
cutter <- function (x, cut = seq(0, 10, by = 2.5)) {
    out <- rep(1, length(x))
    for (i in 2:(length(cut) - 1))
        out[which(x > cut[i] & x <= cut[(i + 1)])] <- i
    return(out)}
## The hierarchy of sample aggregation
levsm <- with(mite.xy, data.frame(
    l2=cutter(y, cut = seq(0, 10, by = 2.5)),
    l3=cutter(y, cut = seq(0, 10, by = 5))))
## Multiplicative diversity partitioning
multipart(mite, levsm, index="renyi", scales=1, nsimul=19)
multipart(mite ~ l2 + l3, levsm, index="renyi", scales=1, nsimul=19)
multipart(mite ~ ., levsm, index="renyi", scales=1, nsimul=19, relative=TRUE)
multipart(mite ~ ., levsm, index="renyi", scales=1, nsimul=19, global=TRUE)
```

---

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

Description

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

Usage

```
nestedchecker(comm)
nestedn0(comm)
nesteddisc(comm, niter = 200)
nestedtemp(comm, ...)
nestednodf(comm, order = TRUE, weighted = FALSE, wbinary = FALSE)
nestedbetasor(comm)
nestedbetajac(comm)
## S3 method for class 'nestedtemp'
plot(x, kind = c("temperature", "incidence"),
    col=rev(heat.colors(100)),  names = FALSE, ...)
## S3 method for class 'nestednodf'
plot(x, col = "red", names = FALSE, ...)
```

Arguments

comm	Community data.
niter	Number of iterations to reorder tied columns.
x	Result object for a plot.
col	Colour scheme for matrix temperatures.
kind	The kind of plot produced.
names	Label columns and rows in the plot using names in comm. If it is a logical vector of length 2, row and column labels are returned accordingly.
order	Order rows and columns by frequencies.
weighted	Use species abundances as weights of interactions.
wbinary	Modify original method so that binary data give the same result in weighted and and unweighted analysis.
...	Other arguments to functions.

Details

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

Function nestedchecker gives the number of checkerboard units, or 2x2 submatrices where both species occur once but on different sites (Stone & Roberts 1990).

Function nestedn0 implements nestedness measure N0 which is the number of absences from the sites which are richer than the most pauperate site species occurs (Patterson & Atmar 1986).

Function nesteddisc implements discrepancy index which is the number of ones that should be shifted to fill a row with ones in a table arranged by species frequencies (Brualdi & Sanderson 1999). The original definition arranges species (columns) by their frequencies, but did not have any method of handling tied frequencies. The nesteddisc function tries to order tied columns to minimize the discrepancy statistic but this is rather slow, and with a large number of tied columns there is no guarantee that the best ordering was found (argument niter gives the maximum number of tried orders). In that case a warning of tied columns will be issued.

Function nestedtemp finds the matrix temperature which is defined as the sum of "surprises" in arranged matrix. In arranged unsurprising matrix all species within proportion given by matrix fill are in the upper left corner of the matrix, and the surprise of the absence or presences is the diagonal distance from the fill line (Atmar & Patterson 1993). Function tries to pack species and sites to a low temperature (Rodríguez-Gironés & Santamaria 2006), but this is an iterative procedure, and the temperatures usually vary among runs. Function nestedtemp also has a plot method which can display either incidences or temperatures of the surprises. Matrix temperature was rather vaguely described (Atmar & Patterson 1993), but Rodríguez-Gironés & Santamaria (2006) are more explicit and their description is used here. However, the results probably differ from other implementations, and users should be cautious in interpreting the results. The details of calculations are explained in the vignette *Design decisions and implementation* that you can read using functions browseVignettes. Function nestedness in the **bipartite** package is a direct port of the BINMATNEST programme of Rodríguez-Gironés & Santamaria (2006).

Function nestednodf implements a nestedness metric based on overlap and decreasing fill (Almeida-Neto et al., 2008). Two basic properties are required for a matrix to have the maximum degree of nestedness according to this metric: (1) complete overlap of 1's from right to left columns and from down to up rows, and (2) decreasing marginal totals between all pairs of columns and all pairs of rows. The nestedness statistic is evaluated separately for columns (N columns) for rows (N rows) and combined for the whole matrix (NODF). If you set order = FALSE, the statistic is evaluated with the current matrix ordering allowing tests of other meaningful hypothesis of matrix structure than default ordering by row and column totals (breaking ties by total abundances when weighted = TRUE) (see Almeida-Neto et al. 2008). With weighted = TRUE, the function finds the weighted version of the index (Almeida-Neto & Ulrich, 2011). However, this requires quantitative null models for adequate testing. Almeida-Neto & Ulrich (2011) say that you have positive nestedness if values in the first row/column are higher than in the second. With this condition, weighted analysis of binary data will always give zero nestedness. With argument wbinary = TRUE, equality of rows/columns also indicates nestedness, and binary data will give identical results in weighted and unweighted analysis. However, this can also influence the results of weighted analysis so that the results may differ from Almeida-Neto & Ulrich (2011).

Functions nestedbetasor and nestedbetajac find multiple-site dissimilarities and decompose these into components of turnover and nestedness following Baselga (2012); the pairwise dissimilarities can be found with designdist. This can be seen as a decomposition of beta diversity (see betadiver). Function nestedbetasor uses Sørensen dissimilarity and the turnover component is Simpson dissimilarity (Baselga 2012), and nestedbetajac uses analogous methods with the Jaccard index. The functions return a vector of three items: turnover, nestedness and their sum which is the multiple Sørensen or Jaccard dissimilarity. The last one is the total beta diversity (Baselga 2012). The functions will treat data as presence/absence (binary) and they can be used with binary nullmodel. The overall dissimilarity is constant in all nullmodels that fix species (column) frequencies ("c0"), and all components are constant if row columns are also fixed (e.g., model "quasiswap"), and the functions are not meaningful with these null models.

Value

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

Author(s)

Jari Oksanen and Gustavo Carvalho (nestednodf).

References

Almeida-Neto, M., Guimarães, P., Guimarães, P.R., Loyola, R.D. & Ulrich, W. (2008). A consistent metric for nestedness analysis in ecological systems: reconciling concept and measurement. *Oikos* 117, 1227–1239.

Almeida-Neto, M. & Ulrich, W. (2011). A straightforward computational approach for measuring nestedness using quantitative matrices. *Env. Mod. Software* 26, 173–178.

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

Baselga, A. (2012). The relationship between species replacement, dissimilarity derived from nestedness, and nestedness. *Global Ecol. Biogeogr.* 21, 1223–1232.

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

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

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

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

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

See Also

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

Examples

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

nobs.cca — *Extract the Number of Observations from a vegan Fit.*

Description

Extract the number of 'observations' from a **vegan** model fit.

Usage

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

Arguments

`object`	A fitted model object.
`...`	Further arguments to be passed to methods.

Details

Function `nobs` is generic in R, and **vegan** provides methods for objects from `betadisper`, `cca` and other related methods, `CCorA`, `decorana`, `isomap`, `metaMDS`, `pcnm`, `procrustes`, `radfit`, `varpart` and `wcmdscale`.

Value

A single number, normally an integer, giving the number of observations.

Author(s)

Jari Oksanen

nullmodel — *Null Model and Simulation*

Description

The `nullmodel` function creates an object which can serve as a basis for Null Model simulation via the `simulate` method. The `update` method updates the nullmodel object without sampling (effective for sequential algorithms). `smbind` binds together multiple `simmat` objects.

Usage

```
nullmodel(x, method)
## S3 method for class 'nullmodel'
print(x, ...)
## S3 method for class 'nullmodel'
simulate(object, nsim = 1, seed = NULL,
    burnin = 0, thin = 1, ...)
## S3 method for class 'nullmodel'
update(object, nsim = 1, seed = NULL, ...)
## S3 method for class 'simmat'
print(x, ...)
smbind(object, ..., MARGIN, strict = TRUE)
```

Arguments

x	A community matrix. For the print method, it is an object to be printed.
method	Character, specifying one of the null model algorithms listed on the help page of commsim. It can be a user supplied object of class commsim.
object	An object of class nullmodel returned by the function nullmodel. In case of smbind it is a simmat object as returned by the update or simulate methods.
nsim	Positive integer, the number of simulated matrices to return. For the update method, it is the number of burnin steps made for sequential algorithms to update the status of the input model object.
seed	An object specifying if and how the random number generator should be initialized ("seeded"). Either NULL or an integer that will be used in a call to set.seed before simulating the matrices. If set, the value is saved as the "seed" attribute of the returned value. The default, NULL will not change the random generator state, and return .Random.seed as the "seed" attribute, see Value.
burnin	Nonnegative integer, specifying the number of steps discarded before starting simulation. Active only for sequential null model algorithms. Ignored for non-sequential null model algorithms.
thin	Positive integer, number of simulation steps made between each returned matrix. Active only for sequential null model algorithms. Ignored for non-sequential null model algorithms.
MARGIN	Integer, indicating the dimension over which multiple simmat objects are to be bound together by smbind. 1: matrices are stacked (row bound), 2: matrices are column bound, 3: iterations are combined. Needs to be of length 1. The other dimensions are expected to match across the objects.
strict	Logical, if consistency of the time series attributes ("start", "end", "thin", and number of simulated matrices) of simmat objects are strictly enforced when binding multiple objects together using smbind. Applies only to input objects based on sequential null model algorithms.
...	Additional arguments supplied to algorithms. In case of smbind it can contain multiple simmat objects.

Details

The purpose of the `nullmodel` function is to create an object, where all necessary statistics of the input matrix are calculated only once. This information is reused, but not recalculated in each step of the simulation process done by the `simulate` method.

The `simulate` method carries out the simulation, the simulated matrices are stored in an array. For sequential algorithms, the method updates the state of the input `nullmodel` object. Therefore, it is possible to do diagnostic tests on the returned `simmat` object, and make further simulations, or use increased thinning value if desired.

The `update` method makes `burnin` steps in case of sequential algorithms to update the status of the input model without any attempt to return matrices. For non-sequential algorithms the method does nothing.

`update` is the preferred way of making burnin iterations without sampling. Alternatively, burnin can be done via the `simulate` method. For convergence diagnostics, it is recommended to use the `simulate` method without burnin. The input nullmodel object is updated, so further samples can be simulated if desired without having to start the process all over again. See Examples.

The `smbind` function can be used to combine multiple `simmat` objects. This comes handy when null model simulations are stratified by sites (`MARGIN = 1`) or by species (`MARGIN = 2`), or in the case when multiple objects are returned by identical/consistent settings e.g. during parallel computations (`MARGIN = 3`). Sanity checks are made to ensure that combining multiple objects is sensible, but it is the user's responsibility to check independence of the simulated matrices and the null distribution has converged in case of sequential null model algorithms. The `strict = FALSE` setting can relax checks regarding start, end, and thinning values for sequential null models.

Value

The function `nullmodel` returns an object of class `nullmodel`. It is a set of objects sharing the same environment:

`data:`	original matrix in integer mode.
`nrow:`	number of rows.
`ncol:`	number of columns.
`rowSums:`	row sums.
`colSums:`	column sums.
`rowFreq:`	row frequencies (number of nonzero cells).
`colFreq:`	column frequencies (number of nonzero cells).
`totalSum:`	total sum.
`fill:`	number of nonzero cells in the matrix.
`commsim:`	the `commsim` object as a result of the `method` argument.
`state:`	current state of the permutations, a matrix similar to the original. It is `NULL` for non-sequential algorithms.
`iter:`	current number of iterations for sequential algorithms. It is `NULL` for non-sequential algorithms.

The simulate method returns an object of class simmat. It is an array of simulated matrices (third dimension corresponding to nsim argument).

The update method returns the current state (last updated matrix) invisibly, and update the input object for sequential algorithms. For non sequential algorithms, it returns NULL.

The smbind function returns an object of class simmat.

Note

Care must be taken when the input matrix only contains a single row or column. Such input is invalid for swapping and several other methods. This also applies to cases when the input is stratified into subsets. In particular, subsetting can generate small or degenerate matrices that cannot be analysed with the selected (or any) null model. These cases are usually detected in commsim and give an error. If you want to handle smoothly error cases, you should wrap simulate in try or tryCatch.

Author(s)

Jari Oksanen and Peter Solymos

See Also

commsim, make.commsim, permatfull, permatswap

Examples

```
data(mite)
x <- as.matrix(mite)[1:12, 21:30]

## non-sequential nullmodel
(nm <- nullmodel(x, "r00"))
(sm <- simulate(nm, nsim=10))

## sequential nullmodel
(nm <- nullmodel(x, "swap"))
(sm1 <- simulate(nm, nsim=10, thin=5))
(sm2 <- simulate(nm, nsim=10, thin=5))

## sequential nullmodel with burnin and extra updating
(nm <- nullmodel(x, "swap"))
(sm1 <- simulate(nm, burnin=10, nsim=10, thin=5))
(sm2 <- simulate(nm, nsim=10, thin=5))

## sequential nullmodel with separate initial burnin
(nm <- nullmodel(x, "swap"))
nm <- update(nm, nsim=10)
(sm2 <- simulate(nm, nsim=10, thin=5))

## combining multiple simmat objects

## stratification
nm1 <- nullmodel(x[1:6,], "r00")
sm1 <- simulate(nm1, nsim=10)
```

```
nm2 <- nullmodel(x[7:12,], "r00")
sm2 <- simulate(nm2, nsim=10)
smbind(sm1, sm2, MARGIN=1)

## binding subsequent samples from sequential algorithms
## start, end, thin retained
nm <- nullmodel(x, "swap")
nm <- update(nm, nsim=10)
sm1 <- simulate(nm, nsim=10, thin=5)
sm2 <- simulate(nm, nsim=20, thin=5)
sm3 <- simulate(nm, nsim=10, thin=5)
smbind(sm3, sm2, sm1, MARGIN=3)

## 'replicate' based usage which is similar to the output
## of 'parLapply' or 'mclapply' in the 'parallel' package
## start, end, thin are set, also noting number of chains
smfun <- function(x, burnin, nsim, thin) {
    nm <- nullmodel(x, "swap")
    nm <- update(nm, nsim=burnin)
    simulate(nm, nsim=nsim, thin=thin)
}
smlist <- replicate(3, smfun(x, burnin=50, nsim=10, thin=5), simplify=FALSE)
smbind(smlist, MARGIN=3) # Number of permuted matrices = 30

## Not run:
## parallel null model calculations
library(parallel)

if (.Platform$OS.type == "unix") {
## forking on Unix systems
smlist <- mclapply(1:3, function(i) smfun(x, burnin=50, nsim=10, thin=5))
smbind(smlist, MARGIN=3)
}

## socket type cluster, works on all platforms
cl <- makeCluster(3)
clusterEvalQ(cl, library(vegan))
clusterExport(cl, c("smfun", "x"))
smlist <- parLapply(cl, 1:3, function(i) smfun(x, burnin=50, nsim=10, thin=5))
stopCluster(cl)
smbind(smlist, MARGIN=3)

## End(Not run)
```

---

oecosimu *Evaluate Statistics with Null Models of Biological Communities*

Description

Function evaluates a statistic or a vector of statistics in community and evaluates its significance in a series of simulated random communities. The approach has been used traditionally for the analysis

of nestedness, but the function is more general and can be used with any statistics evaluated with simulated communities. Function `oecosimu` collects and evaluates the statistics. The Null model communities are described in `make.commsim` and `permatfull`/ `permatswap`, the definition of Null models in `nullmodel`, and nestedness statistics in `nestednodf` (which describes several alternative statistics, including nestedness temperature, $N0$, checker board units, nestedness discrepancy and NODF).

Usage

```
oecosimu(comm, nestfun, method, nsimul = 99, burnin = 0, thin = 1,
   statistic = "statistic", alternative = c("two.sided", "less", "greater"),
   batchsize = NA, parallel = getOption("mc.cores"), ...)
## S3 method for class 'oecosimu'
summary(object, ...)
## S3 method for class 'oecosimu'
as.ts(x, ...)
## S3 method for class 'oecosimu'
toCoda(x)
```

Arguments

`comm`	Community data, or a Null model object generated by `nullmodel` or an object of class `simmat` (array of permuted matrices from `simulate.nullmodel`). If `comm` is a community data, null model simulation `method` must be specified. If `comm` is a `nullmodel`, the simulation `method` is ignored, and if `comm` is a `simmat` object, all other arguments are ignored except `nestfun`, `statistic` and `alternative`.
`nestfun`	Function analysed. Some nestedness functions are provided in **vegan** (see `nestedtemp`), but any function can be used if it accepts the community as the first argument, and returns either a plain number or a vector or the result in list item with the name defined in argument `statistic`. See Examples for defining your own functions.
`method`	Null model method: either a name (character string) of a method defined in `make.commsim` or a `commsim` function. This argument is ignored if `comm` is a `nullmodel` or a `simmat` object. See Details and Examples.
`nsimul`	Number of simulated null communities (ignored if `comm` is a `simmat` object).
`burnin`	Number of null communities discarded before proper analysis in sequential methods (such as `"tswap"`) (ignored with non-sequential methods or when `comm` is a `simmat` object).
`thin`	Number of discarded null communities between two evaluations of nestedness statistic in sequential methods (ignored with non-sequential methods or when `comm` is a `simmat` object).
`statistic`	The name of the statistic returned by `nestfun`.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. Please note that the p-value of two-sided test is approximately two times higher than in the corresponding one-sided test (`"greater"` or `"less"` depending on the sign of the difference).

batchsize	Size in Megabytes of largest simulation object. If a larger structure would be produced, the analysis is broken internally into batches. With default NA the analysis is not broken into batches. See Details.
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package. If you define a nestfun in Windows that needs other R packages than **vegan** or **permute**, you must set up a socket cluster before the call.
x, object	An oecosimu result object.
...	Other arguments to functions.

Details

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

The Null model type can be given as a name (quoted character string) that is used to define a Null model in make.commsim. These include all binary models described by Wright et al. (1998), Jonsson (2001), Gotelli & Entsminger (2003), Miklós & Podani (2004), and some others. There are several quantitative Null models, such those discussed by Hardy (2008), and several that are unpublished (see make.commsim, permatfull, permatswap for discussion). The user can also define her own commsim function (see Examples).

Function works by first defining a nullmodel with given commsim, and then generating a series of simulated communities with simulate.nullmodel. A shortcut can be used for any of these stages and the input can be

1. Community data (comm), Null model function (nestfun) and the number of simulations (nsimul).
2. A nullmodel object and the number of simulations, and argument method is ignored.
3. A three-dimensional array of simulated communities generated with simulate.nullmodel, and arguments method and nsimul are ignored.

The last case allows analysing several statistics with the same simulations.

The function first generates simulations with given nullmodel and then analyses these using the nestfun. With large data sets and/or large number of simulations, the generated objects can be very large, and if the memory is exhausted, the analysis can become very slow and the system can become unresponsive. The simulation will be broken into several smaller batches if the simulated nullmodel objective will be above the set batchsize to avoid memory problems (see object.size for estimating the size of the current data set). The parallel processing still increases the memory needs. The parallel processing is only used for evaluating nestfun. The main load may be in simulation of the nullmodel, and parallel argument does not help there. Function summary is based on summary.permustats and returns information on permutations.

Function as.ts transforms the simulated results of sequential methods into a time series or a ts object. This allows using analytic tools for time series in studying the sequences (see examples). Function toCoda transforms the simulated results of sequential methods into an "mcmc" object of the **coda** package. The **coda** package provides functions for the analysis of stationarity, adequacy of sample size, autocorrelation, need of burn-in and much more for sequential methods, and summary of the results. Please consult the documentation of the **coda** package.

Function permustats provides support to the standard density, qqnorm functions for the simulated values and further graphical support in the **ggvegan** package.

Value

Function oecosimu returns an object of class "oecosimu". The result object has items statistic and oecosimu. The statistic contains the complete object returned by nestfun for the original data. The oecosimu component contains the following items:

statistic	Observed values of the statistic.
simulated	Simulated values of the statistic.
means	Mean values of the statistic from simulations.
z	Standardized effect sizes (SES, a.k.a. the z-values) of the observed statistic based on simulations.
pval	The P-values of the statistic based on simulations.
alternative	The type of testing as given in argument alternative.
method	The method used in nullmodel.
isSeq	TRUE if method was sequential.

Note

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

The internal structure of the function was radically changed in **vegan 2.2-0** with introduction of commsim and nullmodel and deprecation of commsimulator.

Author(s)

Jari Oksanen and Peter Solymos

References

Hardy, O. J. (2008) Testing the spatial phylogenetic structure of local communities: statistical performances of different null models and test statistics on a locally neutral community. *Journal of Ecology* 96, 914–926.

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

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

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

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

See Also

Function `oecosimu` currently defines null models with `commsim` and generates the simulated null model communities with `nullmodel` and `simulate.nullmodel`. For other applications of `oecosimu`, see `treedive` and `raupcrick`.

See also `nestedtemp` (that also discusses other nestedness functions) and `treedive` for another application.

Examples

```
## Use the first eigenvalue of correspondence analysis as an index
## of structure: a model for making your own functions.
data(sipoo)
## Traditional nestedness statistics (number of checkerboard units)
oecosimu(sipoo, nestedchecker, "r0")
## sequential model, one-sided test, a vector statistic
out <- oecosimu(sipoo, decorana, "swap", burnin=100, thin=10,
   statistic="evals", alt = "greater")
out
## Inspect the swap sequence as a time series object
plot(as.ts(out))
lag.plot(as.ts(out))
acf(as.ts(out))
## Use quantitative null models to compare
## mean Bray-Curtis dissimilarities
data(dune)
meandist <- function(x) mean(vegdist(x, "bray"))
mbc1 <- oecosimu(dune, meandist, "r2dtable")
mbc1

## Define your own null model as a 'commsim' function: shuffle cells
## in each row
foo <- function(x, n, nr, nc, ...) {
   out <- array(0, c(nr, nc, n))
   for (k in seq_len(n))
      out[,,k] <- apply(x, 2, function(z) sample(z, length(z)))
   out
}
cf <- commsim("myshuffle", foo, isSeq = FALSE, binary = FALSE,
   mode = "double")
oecosimu(dune, meandist, cf)

## Use pre-built null model
nm <- simulate(nullmodel(sipoo, "curveball"), 99)
oecosimu(nm, nestedchecker)
## Several chains of a sequential model -- this can be generalized
```

```
## for parallel processing (see ?smbind)
nm <- replicate(5, simulate(nullmodel(sipoo, "swap"), 99,
   thin=10, burnin=100), simplify = FALSE)
## nm is now a list of nullmodels: use smbind to combine these into one
## nullmodel with several chains
## IGNORE_RDIFF_BEGIN
nm <- smbind(nm, MARGIN = 3)
nm
oecosimu(nm, nestedchecker)
## IGNORE_RDIFF_END
## After this you can use toCoda() and tools in the coda package to
## analyse the chains (these will show that thin, burnin and nsimul are
## all too low for real analysis).
```

`optspace` *optspace: algorithm for matrix reconstruction from a partially revealed set*

Description

This function was adapted from the original source code in the **Roptspace** R package (version 0.2.3; MIT License) by Raghunandan H. Keshavan, Andrea Montanari, Sewoong Oh (2010). See `ROptSpace::OptSpace` for more information. Let's assume an ideal matrix M with $(m \times n)$ entries with rank r and we are given a partially observed matrix M_E which contains many missing entries. Matrix reconstruction - or completion - is the task of filling in such entries. optspace is an efficient algorithm that reconstructs M from $|E| = O(rn)$ observed elements with relative root mean square error (RMSE)

$$RMSE \leq C(\alpha)\sqrt{nr/|E|}$$

.

Usage

```
optspace(x, ropt = 3, niter = 5, tol = 1e-5, verbose = FALSE)
```

Arguments

`x`	An $(n \times m)$ matrix whose missing entries should be flagged as NA.
`ropt`	`FALSE` to guess the rank, or a positive integer as a pre-defined rank (default: 3).
`niter`	Maximum number of iterations allowed.
`tol`	Stopping criterion for reconstruction in Frobenius norm.
`verbose`	a logical value; `TRUE` to show progress, `FALSE` otherwise.

Details

This implementation removes the trimming step of the original `Roptspace::OptSpace` code in order to leave feature filtering to the user. Some of the defaults have been adjusted to better reflect ecological data. The implementation has been adjusted for ecological applications as in Martino et al. (2019). The imputed matrix (M) in the optspace output includes matrix reconstruction (XSY').

Value

Returns a named list containing:

X	an $(n \times r)$ matrix as left singular vectors.
S	an $(r \times r)$ matrix as singular values.
Y	an $(m \times r)$ matrix as right singular vectors.
dist	a vector containing reconstruction errors at each successive iteration.
M	an $(n \times m)$ imputed matrix as calculated from X, S and Y.

Author(s)

Leo Lahti and Cameron Martino, with adaptations of the method implemented in Roptspace::OptSpace by Keshavan et al. (2010).

References

Keshavan, R. H., Montanari, A., Oh, S. (2010). Matrix Completion From a Few Entries. *IEEE Transactions on Information Theory* **56**:2980–2998. doi:10.1109/TIT.2010.2046205.

Martino, C., Morton, J.T., Marotz, C.A., Thompson, L.R., Tripathi, A., Knight, R. & Zengler, K. (2019) A novel sparse compositional technique reveals microbial perturbations. *mSystems* **4**, 1. doi:10.1128/msystems.0001619.

Examples

```
data(varespec)
# rclr transformation with no matrix completion for the 0/NA entries
x <- decostand(varespec, method = "rclr", impute = FALSE)
# Add matrix completion
xc <- optspace(x, ropt = 3, niter = 5, tol = 1e-5, verbose = FALSE)$M
```

---

ordiarrows *Add Arrows and Line Segments to Ordination Diagrams*

Description

Functions to add arrows, line segments, regular grids of points. The ordination diagrams can be produced by vegan plot.cca, plot.decorana or ordiplot.

Usage

```
ordiarrows(ord, groups, levels, replicates, order.by, display = "sites",
         col = 1, show.groups, startmark, label = FALSE, length = 0.1, ...)
ordisegments(ord, groups, levels, replicates, order.by, display = "sites",
         col = 1, show.groups, label = FALSE, ...)
ordigrid(ord, levels, replicates, display = "sites",  lty = c(1,1),
         col = c(1,1), lwd = c(1,1), ...)
```

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.
`order.by`	Order points by increasing order of this variable within `groups`. Reverse sign of the variable for decreasing ordering.
`display`	Item to displayed.
`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.
`label`	Label the `groups` by their names. In `ordiellipse`, `ordihull` and `ordispider` the the group name is in the centroid of the object, in `ordiarrows` in the start of the arrow, and in `ordisegments` at both ends. `ordiellipse` and `ordihull` use standard `text`, and others use `ordilabel`.
`startmark`	plotting character used to mark the first item. The default is to use no mark, and for instance, `startmark = 1` will draw a circle. For other plotting characters, see `pch` in `points`.
`col`	Colour of lines, `label` borders and `startmark` in `ordiarrows` and `ordisegments`. This can be a vector recycled for `groups`. In `ordigrid` it can be a vector of length 2 used for `levels` and `replicates`.
`length`	Length of edges of the arrow head (in inches).
`lty, lwd`	Line type, line width used for `levels` and `replicates` in `ordigrid`.
`...`	Parameters passed to graphical functions such as `lines`, `segments`, `arrows`, or to `scores` to select axes and scaling etc.

Details

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.

Note

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

Author(s)

Jari Oksanen

See Also

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

Examples

```
example(pyrifos)
mod <- rda(pyrifos)
plot(mod, type = "n")
## Annual succession by ditches, colour by dose
ordiarrows(mod, ditch, label = TRUE, col = as.numeric(dose))
legend("topright", levels(dose), lty=1, col=1:5, title="Dose")
## Show only control and highest Pyrifos treatment
plot(mod, type = "n")
ordiarrows(mod, ditch, label = TRUE,
   show.groups = c("2", "3", "5", "11"))
ordiarrows(mod, ditch, label = TRUE, show = c("6", "9"),
   col = 2)
legend("topright", c("Control", "Pyrifos 44"), lty = 1, col = c(1,2))
```

`ordiArrowTextXY` *Support Functions for Drawing Vectors*

Description

Support functions to assist with drawing of vectors (arrows) on ordination plots. `ordiArrowMul` finds the multiplier for the coordinates of the head of the vector such that they occupy `fill` proportion of the plot region. `ordiArrowTextXY` finds coordinates for the locations of `labels` to be drawn just beyond the head of the vector.

Usage

```
ordiArrowTextXY(x, labels, display, choices = c(1,2),
    rescale = TRUE, fill = 0.75, at = c(0,0), cex = NULL, ...)
ordiArrowMul(x, at = c(0,0), fill = 0.75, display, choices = c(1,2), ...)
```

Arguments

`x`	An R object, from which `scores` can determine suitable ordination scores or an object created by `envfit`, or a two-column matrix of coordinates of arrow heads on the two plot axes.
`labels`	Change plotting labels. A character vector of labels for which label coordinates are sought. If not supplied, these will be determined from the row names of `x`, or `scores(x, ...)` if required. If either of these are not defined, suitable labels will be generated.
`display`	a character string known to `scores` or one of its methods which indicates the type of scores to extract. In fitting functions these are ordinary site scores or linear combination scores (`"lc"`) in constrained ordination (`cca`, `rda`, `dbrda`). If `x` was created by `envfit` then `display` can not be set by the user and takes the value `"vectors"`. Ignored if `x` is a matrix.
`choices`	Axes to be plotted.

rescale	logical; should the coordinates in or extracted from x be rescaled to fill fill proportion of the plot region? The default is to always rescale the coordinates as this is usually desired for objects x from which coordinates are retrieved. If supplying x a 2-column matrix that has already been rescaled, then set this to FALSE.
fill	numeric; the proportion of the plot to fill by the span of the arrows.
at	The origin of fitted arrows in the plot. If you plot arrows in other places than origin, you probably have to specify arrrow.mul.
cex	Character expansion for text.
...	Parameters passed to scores, and strwidth and strheight.

Details

ordiArrowMul finds a multiplier to scale a bunch of arrows to fill an ordination plot, and ordiArrowTextXY finds the coordinates for labels of these arrows. NB., ordiArrowTextXY does not draw labels; it simply returns coordinates at which the labels should be drawn for use with another function, such as text. The arrow will point to the midpoint of the text bounding box. Several ordination plot and text functions have argument optimize which will override this location.

Value

For ordiArrowTextXY, a 2-column matrix of coordinates for the label centres in the coordinate system of the currently active plotting device.

For ordiArrowMul, a length-1 vector containing the scaling factor.

Author(s)

Jari Oksanen, with modifications by Gavin L. Simpson

Examples

```
## Scale arrows by hand to fill 80% of the plot
## Biplot arrows by hand
data(varespec, varechem)
ord <- cca(varespec ~ Al + P + K, varechem)
plot(ord, display = c("species","sites"))

## biplot scores
bip <- scores(ord, choices = 1:2, display = "bp")

## scaling factor for arrows to fill 80% of plot
(mul <- ordiArrowMul(bip, fill = 0.8))
bip.scl <- bip * mul                    # Scale the biplot scores
labs <- rownames(bip)                   # Arrow labels

## calculate coordinate of labels for arrows
(bip.lab <- ordiArrowTextXY(bip.scl, rescale = FALSE, labels = labs))

## arrows will touch the bounding box of the text
```

```
arrows(0, 0, bip.scl[,1], bip.scl[,2], length = 0.1)
ordilabel(bip.lab, labels = labs)

## Handling of ordination objects directly
mul2 <- ordiArrowMul(ord, display = "bp", fill = 0.8)
all.equal(mul, mul2)
```

`ordihull` *Display Groups or Factor Levels in Ordination Diagrams*

Description

Functions to add convex hulls, "spider" graphs, ellipses or cluster dendrogram to ordination diagrams. The ordination diagrams can be produced by vegan `plot.cca`, `plot.decorana` or `ordiplot`.

Usage

```
ordihull(ord, groups, display = "sites", draw = c("lines","polygon", "none"),
         col = NULL, alpha = 0.5, show.groups, label = FALSE,
         border = NULL, lty = NULL, lwd = NULL, ...)
ordiellipse(ord, groups, display="sites", kind = c("sd","se", "ehull"),
         conf, draw = c("lines","polygon", "none"),
  w, col = NULL, alpha = 0.5, show.groups, label = FALSE,
         border = NULL, lty = NULL, lwd=NULL, ...)
ordibar(ord, groups, display = "sites", kind = c("sd", "se"), conf,
         w, col = 1, show.groups, label = FALSE, lwd = NULL, length = 0,  ...)
ordispider(ord, groups, display="sites", w, spiders = c("centroid", "median"),
         show.groups, label = FALSE, col = NULL, lty = NULL, lwd = NULL, ...)
ordicluster(ord, cluster, prune = 0, display = "sites",
            w, col = 1, draw = c("segments", "none"), ...)
## S3 method for class 'ordihull'
summary(object, ...)
## S3 method for class 'ordiellipse'
summary(object, ...)
ordiareatest(ord, groups, area = c("hull", "ellipse"), kind = "sd",
         permutations = 999, parallel = getOption("mc.cores"), ...)
## S3 method for class 'ordiareatest'
summary(object, ...)
```

Arguments

`ord`	An ordination object or an `ordiplot` object.
`groups`	Factor giving the groups for which the graphical item is drawn.
`display`	Item to displayed.

draw — character; how should objects be represented on the plot? For ordihull and ordiellipse use either lines or polygon to draw the lines. For ordicluster, line segments are drawn using segments. To suppress plotting, use "none". Graphical parameters are passed to both. The main difference is that polygons may be filled and non-transparent. With none nothing is drawn, but the function returns the invisible plotting.

col — Colour of hull or ellipse lines (if draw = "lines") or their fills (if draw = "polygon") in ordihull and ordiellipse. When draw = "polygon", the colour of bordering lines can be set with argument border of the polygon function. For other functions the effect depends on the underlining functions this argument is passed to. When multiple values of col are specified these are used for each element of names(table(groups)) (in that order), shorter vectors are recycled. Function ordicluster has no groups, and there the argument will be recycled for points, and the colour of connecting lines is a mixture of point s in the cluster.

alpha — Transparency of the fill colour with draw = "polygon" in ordihull and ordiellipse. The argument takes precedence over possible transparency definitions of the colour. The value must be in range [0, 1], and low values are more 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.

label — Label the groups by their names in the centroid of the object. ordiellipse and ordihull use standard text, and others use ordilabel.

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 — Draw standard deviations of points (sd), standard errors (se) or ellipsoid hulls that enclose all points in the group (ehull).

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.

spiders — Are centres or spider bodies calculated either as centroids (averages) or spatial medians.

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.

object — A result object from ordihull, ordiellipse or ordiareatest. The result is invisible, but it can be saved, and used for summaries (areas etc. of hulls and ellipses).

area — Evaluate the area of convex hulls of ordihull, or of ellipses of ordiellipse.

permutations — a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.

parallel — Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.

lty, lwd, border	Vectors of these parameters can be supplied and will be applied (if appropriate) for each element of names(table(groups)) (in that order). Shorter vectors will be recycled.
length	Width (in inches) of the small ("caps") at the ends of the bar segment (passed to arrows).
...	Parameters passed to other functions.

Details

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

Function ordiellipse draws lines or polygons for ellipses by groups. The function can either draw standard deviation of points (kind="sd") or standard error of the (weighted) centroids (kind="se"), and the (weighted) correlation defines the direction of the principal axis of the ellipse. When kind = "se" is used together with argument conf, the ellipses will show the confidence regions for the locations of group centroids. With kind="ehull" the function draws an ellipse that encloses all points of a group using ellipsoidhull (**cluster** package).

Function ordibar draws crossed "error bars" using either either standard deviation of point scores or standard error of the (weighted) average of scores. These are the principal axes of the corresponding ordiellipse, and are found by principal component analysis of the (weighted) covariance matrix.

Functions ordihull and ordiellipse return invisibly an object that has a summary method that returns the coordinates of centroids and areas of the hulls or ellipses. Function ordiareatest studies the one-sided hypothesis that these areas are smaller than with randomized groups. Argument kind can be used to select the kind of ellipse, and has no effect with convex hulls.

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 corresponding 'LC' score. If the argument is a (invisible) ordihull object, the function will connect the points of the hull to their centroid.

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.

Value

Functions ordihull, ordiellipse and ordispider return the invisible plotting structure.

Function ordispider return the coordinates to which each point is connected (centroids or 'LC' scores).

Function ordihull and ordiellipse return invisibly an object that has a summary method that returns the coordinates of centroids and areas of the hulls or ellipses. Function ordiareatest studies the one-sided hypothesis that these areas are smaller than with randomized groups, and its summary is based on summary.permustats with a summary of permutations.

Note

These functions add graphical items to ordination graph: You must draw a graph first. To draw line segments, grids or arrows, see ordisegments, ordigrid andordiarrows.

Author(s)

Jari Oksanen with improvements by James Weedon

See Also

The functions pass parameters to basic graphical functions, and you may wish to change the default values in lines, segments and polygon. You can pass parameters to scores as well. Underlying functions for ordihull is chull. The underlying function for ellipsoid hulls in ordiellipse is ellipsoidhull.

Examples

```
data(dune)
data(dune.env)
mod <- cca(dune ~ Management, dune.env)
plot(mod, type="n", scaling = "symmetric")
## Catch the invisible result of ordihull...
pl <- with(dune.env, ordihull(mod, Management,
                              scaling = "symmetric", label = TRUE))
## ... and find centres and areas of the hulls
summary(pl)
## use more colours and add ellipsoid hulls
plot(mod, type = "n")
pl <- with(dune.env, ordihull(mod, Management,
                              scaling = "symmetric", col = 1:4,
                              draw="polygon", label =TRUE))
with(dune.env, ordiellipse(mod, Management, scaling = "symmetric",
                           kind = "ehull", col = 1:4, lwd=3))
## ordispider to connect WA and LC scores
plot(mod, dis=c("wa","lc"), type="p")
ordispider(mod)
## Other types of plots
plot(mod, type = "p", display="sites")
cl <- hclust(vegdist(dune))
ordicluster(mod, cl, prune=3, col = cutree(cl, 4))
## confidence ellipse: location of the class centroids
plot(mod, type="n", display = "sites")
with(dune.env, text(mod, display="sites", labels = as.character(Management),
                    col=as.numeric(Management)))
pl <- with(dune.env, ordiellipse(mod, Management, kind="se", conf=0.95, lwd=2,
                                 draw = "polygon", col=1:4, border=1:4,
                                 alpha=63))
summary(pl)
## add confidence bars
with(dune.env, ordibar(mod, Management, kind="se", conf=0.95, lwd=2, col=1:4,
                       label=TRUE))
```

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

Description

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

Usage

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

Arguments

x	An ordination object an any object known to scores.
display	Kind of scores displayed (passed to scores).
labels	Optional text used in plots instead of the default. If select is given, the labels are given only to selected items in the order they occur in the scores.
choices	Axes shown (passed to scores).
priority	Vector of the same length as the number of scores or selected items. The items with high priority will be plotted uppermost.
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.
cex	Character expansion for the text (passed to text).
fill	Background colour of the labels (the col argument of polygon).
border	The colour and visibility of the border of the label as defined in polygon. The default is to use text colour col.
col	Text colour.
xpd	Draw labels also outside the plot region.
...	Other arguments (passed to text).

Details

The function may be useful with crowded ordination plots, in particular together with argument priority. You will not see all text labels, but at least some are readable. Function can be used as a part of a pipe (|>) in place of text after an ordination plot command (see Examples).

Other alternatives for cluttered plots are identify.ordiplot, orditorp, ordipointlabel, and orditkplot (**vegan3d** package).

Author(s)

Jari Oksanen

See Also

plot.cca and text.ordiplot that can use the function with argument bg.

Examples

```
data(dune)
ord <- cca(dune)
plot(ord, type = "n")
## add text
ordilabel(ord, dis="sites", cex=1.2, font=3, fill="hotpink", col="blue")
## You may prefer separate plots, but here species as well
ordilabel(ord, dis="sp", font=2, priority=colSums(dune))
## use in a pipe
plot(ord, type = "n") |>
   ordilabel("spec", font = 3, priority = colSums(dune)) |>
   points("sites", pch=21, bg = "yellow", col = "blue")
```

---

ordiplot — *Alternative plot and identify Functions for Ordination*

Description

Function ordiplot is an alternative plotting function which works with any **vegan** ordination object and many non-**vegan** objects. In addition, plot functions for **vegan** ordinations return invisibly an "ordiplot" object, and this allows using ordiplot support functions with this result: identify can be used to add labels to selected site, species or constraint points, and points and text can add elements to the plot, and used in a pipe to add scores into plot by layers.

Usage

```
ordiplot(ord, choices = c(1, 2), type="points", display, optimize = FALSE,
    arrows = FALSE, length = 0.05, arr.mul, xlim, ylim, ...)
## S3 method for class 'ordiplot'
points(x, what, select, arrows = FALSE,
    length = 0.05, arr.mul, ...)
## S3 method for class 'ordiplot'
text(x, what, labels, select, optimize = FALSE,
    arrows = FALSE, length = 0.05, arr.mul, bg, ...)
## S3 method for class 'ordiplot'
identify(x, what, labels,  ...)
```

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, dbrda 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, biplot and regression, and plot.procrustes ordination plot has heads and points.
labels	Optional text used for labels. Row names of scores will be used if this is missing. If select is used, labels are given only to selected items in the order they occur in the scores.
optimize	Optimize locations of text to reduce overlap and plot point in the actual locations of the scores. Uses ordipointlabel.
arrows	Draw arrows from the origin. This will always be TRUE for biplot and regression scores in constrained ordination (cca etc.). Setting this TRUE will draw arrows for any type of scores. This allows, e.g, using biplot arrows for species. The arrow head will be at the value of scores, and possible text is moved outwards.
length	Length of arrow heads (see arrows).
arr.mul	Numeric multiplier to arrow lengths; this will also set arrows = TRUE. The default is to automatically adjust arrow lengths with "biplot" and "regression" scores and else use unmodified scores.
bg	Background colour for labels. If bg is set, the labels are displayed with ordilabel instead of text.
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 with default of black circles for sites and red crosses for species. It returns invisibly an object of class ordiplot.

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. Alternatively, points and text can

be used in pipe which allows an intuitive way of building up plots by layers. In addition, function ordilabel and ordipointlabel can be used in pipe after ordiplot or other **vegan** ordination plot commands. See Examples.

Value

Function ordiplot returns invisibly an object of class ordiplot with used scores. In general, **vegan** plot functions for ordination results will also return an invisible ordiplot object. If the plot(..., type = "n") was used originally, the plot is empty, and items can be added with the invisible object. Functions points and text return their input object without modification, which allows chaining these commands with pipes. Function identify.ordiplot uses this object to label the point.

Author(s)

Jari Oksanen

See Also

With argument bg function calls ordilabel to draw text on non-transparent label, and with argument optimize = TRUE function calls ordipointlabel to optimize the locations of text labels to minimize over-plotting. Functions ordilabel and ordipointlabel can be used in a pipe together with ordiplot methods text and points. Function plot.cca uses ordiplot methods text and points in configurable plots, and these accept the arguments of the ordiplot methods described here.

Examples

```
## Draw a plot for a non-vegan ordination (stats:::cmdscale).
data(dune)
dune.dis <- vegdist(wisconsin(dune))
dune.mds <- cmdscale(dune.dis, eig = TRUE)
dune.mds$species <- wascores(dune.mds$points, dune, expand = TRUE)
pl <- ordiplot(dune.mds, type = "none")
points(pl, "sites", pch=21, col="red", bg="yellow")
text(pl, "species", col="blue", cex=0.9)
## stats::prcomp biplot using pipes
pc <- prcomp(USArrests, scale=TRUE) # cf example(prcomp)
ordiplot(pc, type = "n") |>
  text("sites", optimize=TRUE, xpd=TRUE) |>
  text("species", arr.mul=2.5, col=2)

## same plot using pipes (|>)
ordiplot(dune.mds, type="n") |>
  points("sites", pch=21, col="red", bg="yellow") |>
  text("species", col="blue", cex=0.9)
## Default plot using identify to label selected points
## Not run:
pl <- ordiplot(dune.mds)
identify(pl, "spec")
## End(Not run)
```

```
## Some people think that species should be shown with arrows in PCA.
## Other ordination methods also return an invisible ordiplot object and
## we can use pipes to draw those arrows. We drop species that are
## poorly explained by the current graph.
mod <- rda(dune)
good <- goodness(mod, choices=1:2, model="CA", summarize = TRUE)
plot(mod, type="n") |>
  points("sites", pch=16, col="red") |>
  text("species", arrows = TRUE, length=0.05, col="blue", select = good > 1/3)
```

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

Description

Function produces ordination plots with labelled points. The points are in the fixed locations given by the ordination, but the locations of the text labels are optimized to minimize overplotting. The function is useful with moderately crowded ordination plots.

Usage

```
ordipointlabel(x, display = c("sites", "species"), choices = c(1, 2),
   col = c(1, 2),  pch = c("o", "+"), font = c(1, 1),
   cex = c(0.7, 0.7), add = inherits(x, "ordiplot"), labels, bg, select,
   points = TRUE, ...)

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

Arguments

`x`	For `ordipointlabel` a result object from an ordination function or an ordination plot (possibly in a pipe). For `plot.ordipointlabel` an object from `ordipointlabel`.
`display`	Scores displayed in the plot. The default is to show `"sites"` and `"species"` that are available to many ordination methods, but there can be only one set or more than two set of scores to `display`.
`choices`	Axes shown.
`col`, `pch`, `font`, `cex`	Colours, point types, font style and character expansion for each kind of scores displayed in the plot. These should be vectors of the same length as the number of items in `display`, or if there is only one `display` they can be a vector of the length of number items.
`add`	Add to an existing plot. Default is `add = TRUE` when the function is used in a pipe, and `FALSE` usually.

labels	Labels used in graph. Species (variable) and SU (row) names are used if this is missing. Labels must be given in one vector for all scores of display. Function labels can extract the current name from a saved ordipointlabel object. If select is used, labels are given only for the selected items.
bg	Background colour for labels. If this is given, texts is drawn over non-transparent background. Either a single colour or vector of colours for each display, or with one display, for each label.
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. select is only used if a single set of scores is being plotted (i.e. length(display) == 1), otherwise it is ignored and a warning issued. If a logical vector is used, it must have the same length as the scores plotted.
points	Set to FALSE to suppress drawing points. This is useful if points (or arrowheads) already exist.
...	Other arguments passed to points and text.

Details

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

The function can be used in a pipe where the first command is an ordination plot command with type = "n" or to add points and labels to save **vegan** ordination plot object. See examples.

Value

The function returns invisibly an object of class ordipointlabel with items xy for coordinates of points, labels for coordinates of labels, items pch, cex and font for graphical parameters of each point or label. In addition, it returns the result of optim as an attribute "optim". The unit of overlap is the area of character "m", and with varying graphical parameters the smallest alternative.

There is a plot method based on orditkplot but it does not alter or reset the graphical parameters via par.

The result object from ordipointlabel is similar as the orditkplot object of **vegan3d** package, and it may be possible to further edit the result object with orditkplot, but for good results it is necessary that the points span the whole horizontal axis without empty margins.

Author(s)

Jari Oksanen

See Also

The function is invoked for one set of scores (one display) from text.ordiplot and plot.cca with argument optimize = TRUE.

Examples

```
data(dune, dune.env)
ord <- cca(dune)
ordipointlabel(ord)

## Use in a pipe: optimize species, sites & centroids together
ord <- cca(dune ~ Management + Moisture, dune.env)
plot(ord, scaling = "symmetric", type = "n") |>
  ordipointlabel(c("sites","species","centroids"), cex=c(0.7,0.7,1),
      col = c("black","red","blue"), font = c(1,3,1), pch=c(1,3,4), xpd=TRUE) |>
  text("biplot", col = "blue", bg = "white", cex=1)
```

`ordistep` *Choose a Model by Permutation Tests in Constrained Ordination*

Description

Automatic stepwise model building for constrained ordination methods (`cca`, `rda`, `dbrda`, `capscale`). The function `ordistep` is modelled after `step` and can do forward, backward and stepwise model selection using permutation tests. Function `ordiR2step` performs forward model choice solely on adjusted R^2 and P-value.

Usage

```
ordistep(object, scope, direction = c("both", "backward", "forward"),
  Pin = 0.05, Pout = 0.1, permutations = how(nperm = 199), steps = 50,
  trace = TRUE, test = c("permutation", "F"), ...)
ordiR2step(object, scope, Pin = 0.05, R2scope = TRUE,
  permutations = how(nperm = 499), trace = TRUE, R2permutations = 1000,
  test = c("permutation", "F"),  ...)
```

Arguments

`object`	In `ordistep`, an ordination object inheriting from `cca` or `rda`.
`scope`	Defines the range of models examined in the stepwise search. This can be a list containing components `upper` and `lower`, both formulae. If it is a single item, it is interpreted the target scope, depending on the `direction`. If `direction` is `"forward"`, a single item is interpreted as the `upper` scope and the formula of the input `object` as the `lower` scope. See `step` for details. In `ordiR2step`, this defines the upper scope; it can also be an ordination object from with the model is extracted.
`direction`	The mode of stepwise search, can be one of `"both"`, `"backward"`, or `"forward"`, with a default of `"both"`. If the `scope` argument is missing, the default for `direction` is `"backward"` in `ordistep` (and `ordiR2step` does not have this argument, but only works forward).

Pin, Pout	Limits of permutation P-values for adding (Pin) a term to the model, or dropping (Pout) from the model. Term is added if $P \leq$ Pin, and removed if $P >$ Pout.
R2scope	Use adjusted R^2 as the stopping criterion: only models with lower adjusted R^2 than scope are accepted.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices. This is passed to anova.cca: see there for details.
steps	Maximum number of iteration steps of dropping and adding terms.
trace	If positive, information is printed during the model building. Larger values may give more information.
R2permutations	Number of permutations used in the estimation of adjusted R^2 for cca using RsquareAdj.
test	Use "permutation" test or parametric "F" test in anova.cca. "F" can be only used with rda models.
...	Any additional arguments to add1.cca and drop1.cca.

Details

The basic functions for model choice in constrained ordination are add1.cca and drop1.cca. With these functions, ordination models can be chosen with standard R function step which bases the term choice on AIC. AIC-like statistics for ordination are provided by functions deviance.cca and extractAIC.cca (with similar functions for rda). Actually, constrained ordination methods do not have AIC, and therefore the step may not be trusted. This function provides an alternative using permutation P-values.

Function ordistep defines the model, scope of models considered, and direction of the procedure similarly as step. The function alternates with drop and add steps and stops when the model was not changed during one step. The - and + signs in the summary table indicate which stage is performed. It is often sensible to have Pout $>$ Pin in stepwise models to avoid cyclic adds and drops of single terms.

Function ordiR2step builds model forward so that it maximizes adjusted R^2 (function RsquareAdj) at every step, and stopping when the adjusted R^2 starts to decrease, or the adjusted R^2 of the scope is exceeded, or the selected permutation P-value is exceeded (Blanchet et al. 2008). The second criterion is ignored with option R2scope = FALSE, and the third criterion can be ignored setting Pin = 1 (or higher). The function cannot be used if adjusted R^2 cannot be calculated. If the number of predictors is higher than the number of observations, adjusted R^2 is also unavailable. Such models can be analysed with R2scope = FALSE, but the variable selection will stop if models become overfitted and adjusted R^2 cannot be calculated, and the adjusted R^2 will be reported as zero. The R^2 of cca is based on simulations (see RsquareAdj) and different runs of ordiR2step can give different results.

Functions ordistep (based on P values) and ordiR2step (based on adjusted R^2 and hence on eigenvalues) can select variables in different order.

Value

Functions return the selected model with one additional component, `anova`, which contains brief information of steps taken. You can suppress voluminous output during model building by setting `trace = FALSE`, and find the summary of model history in the `anova` item.

Author(s)

Jari Oksanen

References

Blanchet, F. G., Legendre, P. & Borcard, D. (2008) Forward selection of explanatory variables. *Ecology* 89, 2623–2632.

See Also

The function handles constrained ordination methods `cca`, `rda`, `dbrda` and `capscale`. The underlying functions are `add1.cca` and `drop1.cca`, and the function is modelled after standard `step` (which also can be used directly but uses AIC for model choice, see `extractAIC.cca`). Function `ordiR2step` builds upon `RsquareAdj`.

Examples

```
## See add1.cca for another example

### Dune data
data(dune)
data(dune.env)
mod0 <- rda(dune ~ 1, dune.env)  # Model with intercept only
mod1 <- rda(dune ~ ., dune.env)  # Model with all explanatory variables

## With scope present, the default direction is "both"
mod <- ordistep(mod0, scope = formula(mod1))
mod
## summary table of steps
mod$anova

## Example of ordistep, forward
ordistep(mod0, scope = formula(mod1), direction="forward")

## Example of ordiR2step (always forward)
## stops because R2 of 'mod1' exceeded
ordiR2step(mod0, mod1)
```

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

Description

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

Usage

```
## Default S3 method:
ordisurf(x, y, choices = c(1, 2),
    what = c("contour", "surface"), knots = 10,
         family = "gaussian", col = "red", isotropic = TRUE,
         bs = "tp", fx = FALSE, add = FALSE, display = "sites", w, main,
         nlevels = 10, levels, npoints = 51, labcex = 0.6,
         bubble = FALSE, cex = 1, select = TRUE, method = "REML",
         gamma = 1, plot = TRUE, lwd.cl = par("lwd"), ...)

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

## S3 method for class 'ordisurf'
calibrate(object, newdata, ...)

## S3 method for class 'ordisurf'
plot(x, what = c("contour","surface", "persp","gam"),
     add = FALSE, bubble = FALSE, col = "red", alpha = 1, cex = 1,
     nlevels = 10, levels, labcex = 0.6, lwd.cl = par("lwd"), ...)
```

Arguments

x	For ordisurf an ordination configuration, either a matrix or a result known by scores. For plot.ordisurf an object of class "ordisurf" as returned by ordisurf.
y	Variable to be plotted / modelled as a function of the ordination scores.
choices	Ordination axes.
what	What type of plot to produce. "contour" produces a contour plot of the response surface, see contour for details. "surface" produces a coloured image surface with contours; since image can cover earlier items, you should set it transparent with alpha parameter, or draw it before other items. "persp" produces a perspective plot of the same, see persp for details. "gam" plots the fitted GAM model, an object that inherits from class "gam" returned by ordisurf, see plot.gam.

knots	Number of initial knots in gam (one more than degrees of freedom). If knots = 0 or knots = 1 the function will fit a linear trend surface, and if knots = 2 the function will fit a quadratic trend surface instead of a smooth surface. A vector of length 2 is allowed when isotropic = FALSE, with the first and second elements of knots referring to the first and second of ordination dimensions (as indicated by choices) respectively.
family	Error distribution in gam.
col	Colour of contours. This can also be used to set the colour palette in what = "surface" or "persp".
isotropic	Fit an isotropic smooth surface (i.e. same smoothness in both ordination dimensions) via gam.
bs	a two letter character string indicating the smoothing basis to use. (e.g. "tp" for thin plate regression spline, "cr" for cubic regression spline). One of c("tp", "ts", "cr", "cs", "ds", "ps", "ad"). See smooth.terms for an over view of what these refer to. The default is to use thin plate splines: bs = "tp".
fx	indicates whether the smoothers are fixed degree of freedom regression splines (fx = FALSE) or penalised regression splines (fx = TRUE). Can be a vector of length 2 for anisotropic surfaces (isotropic = FALSE). It doesn't make sense to use fx = TRUE **and** select = TRUE and it is an **error** to do so. A warning is issued if you specify fx = TRUE and forget to use select = FALSE though fitting continues using select = FALSE.
add	Add contours to 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. Weights of the ordination object will be used if the object has attribute weights or a weights function. Concerns mainly cca and decorana results which have nonconstant weights.
main	The main title for the plot, or as default the name of plotted variable in a new plot.
nlevels, levels	Either a vector of levels for which contours are drawn, or suggested number of contours in nlevels if levels are not supplied.
npoints	numeric; the number of locations at which to evaluate the fitted surface. This represents the number of locations in each dimension.
labcex	Label size in contours. Setting this zero will suppress labels.
bubble	Use a "bubble plot" for points, or vary the point diameter by the value of the plotted variable. If bubble is numeric, its value is used for the maximum symbol size (as in cex), or if bubble = TRUE, the value of cex gives the maximum. The minimum size will always be cex = 0.4. The option only has an effect if add = FALSE.
cex	Character expansion of plotting symbols.
select	Logical; specify gam argument "select". If this is TRUE then gam can add an extra penalty to each term so that it can be penalized to zero. This means that the smoothing parameter estimation that is part of fitting can completely remove terms from the model. If the corresponding smoothing parameter is estimated as zero then the extra penalty has no effect.

method	character; the smoothing parameter estimation method. Options allowed are: "GCV.Cp" uses GCV for models with unknown scale parameter and Mallows' Cp/UBRE/AIC for models with known scale; "GACV.Cp" as for "GCV.Cp" but uses GACV (Generalised Approximate CV) instead of GCV; "REML" and "ML" use restricted maximum likelihood or maximum likelihood estimation for both known and unknown scale; and "P-REML" and "P-ML" use REML or ML estimation but use a Pearson estimate of the scale.
gamma	Multiplier to inflate model degrees of freedom in GCV or UBRE/AIC score by. This effectively places an extra penalty on complex models. An oft-used value is gamma = 1.4.
plot	logical; should any plotting be done by ordisurf? Useful if all you want is the fitted response surface model.
lwd.cl	numeric; the lwd (line width) parameter to use when drawing the contour lines.
formula, data	Alternative definition of the fitted model as x ~ y, where left-hand side is the ordination x and right-hand side the single fitted continuous variable y. The variable y must be in the working environment or in the data frame or environment given by data. All other arguments of are passed to the default method.
object	An ordisurf result object.
newdata	Coordinates in two-dimensional ordination for new points.
alpha	An alpha-transparency level in the range [0,1] (0 means transparent and 1 means opaque) for coloured surface.
...	Other parameters passed to scores, or to the graphical functions. See Note below for exceptions.

Details

Function ordisurf fits a smooth surface using penalised splines (Wood 2003) in gam, and uses predict.gam to find fitted values in a regular grid. The smooth surface can be fitted with an extra penalty that allows the entire smoother to be penalized back to 0 degrees of freedom, effectively removing the term from the model (see Marra & Wood, 2011). The addition of this extra penalty is invoked by setting argument select to TRUE. An alternative is to use a spline basis that includes shrinkage (bs = "ts" or bs = "cs").

ordisurf() exposes a large number of options from gam for specifying the basis functions used for the surface. If you stray from the defaults, do read the **Notes** section below and relevant documentation in s and smooth.terms.

The function plots the fitted contours with convex hull of data points either over an existing ordination diagram or draws a new plot. If select = TRUE and the smooth is effectively penalised out of the model, no contours will be plotted.

gam determines the degree of smoothness for the fitted response surface during model fitting, unless fx = TRUE. Argument method controls how gam performs this smoothness selection. See gam for details of the available options. Using "REML" or "ML" yields p-values for smooths with the best coverage properties if such things matter to you.

The function uses scores to extract ordination scores, and x can be any result object known by that function.

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

Function calibrate returns the fitted values of the response variable. The newdata must be coordinates of points for which the fitted values are desired. The function is based on predict.gam and will pass extra arguments to that function.

Value

ordisurf is usually called for its side effect of drawing the contour plot. The function returns a result object of class "ordisurf" that inherits from gam used internally to fit the surface, but adds an item grid that contains the data for the grid surface. The item grid has elements x and y which are vectors of axis coordinates, and element z that is a matrix of fitted values for contour. The values outside the convex hull of observed points are indicated as NA in z. The gam component of the result can be used for further analysis like predicting new values (see predict.gam).

Warning

The fitted GAM is a regression model and has the usual assumptions of such models. Of particular note is the assumption of independence of residuals. If the observations are not independent (e.g. they are repeat measures on a set of objects, or from an experimental design, *inter alia*) do not trust the *p*-values from the GAM output.

If you need further control (i.e. to add additional fixed effects to the model, or use more complex smoothers), extract the ordination scores using the scores function and then generate your own gam call.

Note

The default is to use an isotropic smoother via s employing thin plate regression splines (bs = "tp"). These make sense in ordination as they have equal smoothing in all directions and are rotation invariant. However, if different degrees of smoothness along dimensions are required, an anisotropic smooth surface may be more applicable. This can be achieved through the use of isotropic = FALSE, wherein the surface is fitted via a tensor product smoother via te (unless bs = "ad", in which case separate splines for each dimension are fitted using s).

Cubic regression splines and P splines can **only** be used with isotropic = FALSE.

Adaptive smooths (bs = "ad"), especially in two dimensions, require a large number of observations; without many hundreds of observations, the default complexities for the smoother will exceed the number of observations and fitting will fail.

Graphical arguments supplied to plot.ordisurf are passed on to the underlying plotting functions, contour, image, persp, and plot.gam. The exception to this is that arguments col and cex can not currently be passed to plot.gam.

A work-around is to call plot.gam directly on the result of a call to ordisurf. See the Examples for an illustration of this.

Author(s)

Dave Roberts, Jari Oksanen and Gavin L. Simpson

References

Marra, G.P & Wood, S.N. (2011) Practical variable selection for generalized additive models. *Comput. Stat. Data Analysis* 55, 2372–2387.

Wood, S.N. (2003) Thin plate regression splines. *J. R. Statist. Soc. B* 65, 95–114.

See Also

For basic routines `gam`, and `scores`. Function `envfit` provides a more traditional and compact alternative. If the fitted surface simplifies to a plane, `envfit` arrowsshow the direction of the steepest gradient.

Examples

```
data(varespec, varechem)
vare.mds <- metaMDS(varespec, trace = 0)
## IGNORE_RDIFF_BEGIN
fit <- ordisurf(vare.mds ~ Baresoil, varechem, bubble = 5)
## fitted values
fitted(fit) # or predict(fit)

## Cover of reindeer lichen Cladonia stellaris
fit <- ordisurf(vare.mds ~ Cladstel, varespec, family=quasipoisson,
    what = "surface")
## Gaussian response function
ordisurf(vare.mds ~ Cladstel, varespec, family=quasipoisson, knots = 2,
    main = "Cladstel: Gaussian Response Surface")

## Variable selection via additional shrinkage penalties
## This allows non-significant smooths to be selected out
## of the model not just to a linear surface. There are 2
## options available:
##  - option 1: `select = TRUE` --- the *default*
ordisurf(vare.mds ~ Baresoil, varechem, method = "REML", select = TRUE)
##  - option 2: use a basis with shrinkage
ordisurf(vare.mds ~ Baresoil, varechem, method = "REML", bs = "ts")
## or bs = "cs" with `isotropic = FALSE`
## IGNORE_RDIFF_END
## Plot method
plot(fit, what = "persp", col = terrain.colors(50),
    main = "Cladstel: GAM Surface")

## Plotting the "gam" object
plot(fit, what = "gam") ## 'col' and 'cex' not passed on
## or via plot.gam directly
library(mgcv)
plot.gam(fit, cex = 2, pch = 1, col = "blue")
## 'col' effects all objects drawn...
```

```
### controlling the basis functions used
## Use Duchon splines
ordisurf(vare.mds ~ Baresoil, varechem, bs = "ds")

## A fixed degrees of freedom smooth, must use 'select = FALSE'
ordisurf(vare.mds ~ Baresoil, varechem, knots = 4,
                        fx = TRUE, select = FALSE)

## An anisotropic smoother with cubic regression spline bases
ordisurf(vare.mds ~ Baresoil, varechem, isotropic = FALSE,
                        bs = "cr", knots = 4)

## An anisotropic smoother with cubic regression spline with
## shrinkage bases & different degrees of freedom in each dimension
ordisurf(vare.mds ~ Baresoil, varechem, isotropic = FALSE,
                        bs = "cs", knots = c(3,4), fx = TRUE,
                        select = FALSE)
```

`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,
    select, cex = 0.7, pcex, col = par("col"), pcol,
    pch = par("pch"), air = 1, ...)
```

Arguments

`x`	A result object from ordination or an `ordiplot` result. If the function is used in `ordiplot` pipe, this should be missing and first argument be `display`.
`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 of scores will be used if this is missing. If `select` is used, labels are given only selected items in the order they occur in the scores.
`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 or number of scores.

select	Items to be displayed. This can either be a logical vector which is TRUE for displayed items or a vector of indices, or labels and if these are missing, row names of scores. If a logical vector is used, it must have the same length as the scores plotted.
cex, pcex	Text and point sizes, see plot.default..
col, pcol	Text and point colours, see plot.default.
pch	Plotting character, see points.
air	Amount of empty space between text labels. Values <1 allow overlapping text.
...	Other arguments to scores (and its various methods), text and points.

Details

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

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

Value

The function returns invisibly the 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. If function is used in an ordiplot pipe, it will return the input ordiplot object, but amend the plotted scores with this vector as attribute "orditorp". 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", labels = cnam, priority=stems, pch="+", pcol="grey")

## show select in action
set.seed(1)
take <- sample(ncol(BCI), 50)
```

```
plot(mod, dis="sp", type="n")
stems <- colSums(BCI)
## only selected items are labelled, and the labels must be in the some
## order as in the scores
orditorp(mod, "sp", labels = cnam[sort(take)], priority=stems, select = take,
         pch="+", pcol="grey")
```

---

pcnm *Principal Coordinates of Neighbourhood Matrix*

Description

This function computed classical PCNM by the principal coordinate analysis of a truncated distance matrix. These are commonly used to transform (spatial) distances to rectangular data that suitable for constrained ordination or regression.

Usage

```
pcnm(dis, threshold, w, dist.ret = FALSE)
```

Arguments

dis	A distance matrix.
threshold	A threshold value or truncation distance. If missing, minimum distance giving connected network will be used. This is found as the longest distance in the minimum spanning tree of dis.
w	Prior weights for rows.
dist.ret	Return the distances used to calculate the PCNMs.

Details

Principal Coordinates of Neighbourhood Matrix (PCNM) map distances between rows onto rectangular matrix on rows using a truncation threshold for long distances (Borcard & Legendre 2002). If original distances were Euclidean distances in two dimensions (like normal spatial distances), they could be mapped onto two dimensions if there is no truncation of distances. Because of truncation, there will be a higher number of principal coordinates. The selection of truncation distance has a huge influence on the PCNM vectors. The default is to use the longest distance to keep data connected. The distances above truncation threshold are given an arbitrary value of 4 times threshold. For regular data, the first PCNM vectors show a wide scale variation and later PCNM vectors show smaller scale variation (Borcard & Legendre 2002), but for irregular data the interpretation is not as clear.

The PCNM functions are used to express distances in rectangular form that is similar to normal explanatory variables used in, e.g., constrained ordination (rda, cca and dbrda) or univariate regression (lm) together with environmental variables (row weights should be supplied with cca; see Examples). This is regarded as a more powerful method than forcing rectangular environmental

data into distances and using them in partial mantel analysis (mantel.partial) together with geographic distances (Legendre et al. 2008, but see Tuomisto & Ruokolainen 2008).

The function is based on pcnm function in Dray's unreleased **spacemakeR** package. The differences are that the current function uses spantree as an internal support function. The current function also can use prior weights for rows by using weighted metric scaling of wcmdscale. The use of row weights allows finding orthonormal PCNMs also for correspondence analysis (e.g., cca).

Value

A list of the following elements:

values	Eigenvalues obtained by the principal coordinates analysis.
vectors	Eigenvectors obtained by the principal coordinates analysis. They are scaled to unit norm. The vectors can be extracted with scores function. The default is to return all PCNM vectors, but argument choices selects the given vectors.
threshold	Truncation distance.
dist	The distance matrix where values above threshold are replaced with arbitrary value of four times the threshold. String "pcnm" is added to the method attribute, and new attribute threshold is added to the distances. This is returned only when dist.ret = TRUE.

Author(s)

Jari Oksanen, based on the code of Stephane Dray.

References

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

Legendre, P., Borcard, D and Peres-Neto, P. (2008) Analyzing or explaining beta diversity? Comment. *Ecology* **89**, 3238–3244.

Tuomisto, H. & Ruokolainen, K. (2008) Analyzing or explaining beta diversity? A reply. *Ecology* **89**, 3244–3256.

See Also

spantree.

Examples

```
## Example from Borcard & Legendre (2002)
data(mite.xy)
pcnm1 <- pcnm(dist(mite.xy))
op <- par(mfrow=c(1,3))
## Map of PCNMs in the sample plot
ordisurf(mite.xy, scores(pcnm1, choi=1), bubble = 4, main = "PCNM 1")
ordisurf(mite.xy, scores(pcnm1, choi=2), bubble = 4, main = "PCNM 2")
ordisurf(mite.xy, scores(pcnm1, choi=3), bubble = 4, main = "PCNM 3")
par(op)
```

```
## Weighted PCNM for CCA
data(mite)
rs <- rowSums(mite)/sum(mite)
pcnmw <- pcnm(dist(mite.xy), w = rs)
ord <- cca(mite ~ scores(pcnmw))
## Multiscale ordination: residual variance should have no distance
## trend
msoplot(mso(ord, mite.xy))
```

---

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

Description

Individual (for count data) or incidence (for presence-absence data) based null models can be generated for community level simulations. Options for preserving characteristics of the original matrix (rows/columns sums, matrix fill) and restricted permutations (based on strata) are discussed in the Details section.

Usage

```
permatfull(m, fixedmar = "both", shuffle = "both", strata = NULL,
    mtype = "count", times = 99, ...)
permatswap(m, method = "quasiswap", fixedmar="both", shuffle = "both",
    strata = NULL, mtype = "count", times = 99,
    burnin = 0, thin = 1, ...)
## S3 method for class 'permat'
print(x, digits = 3, ...)
## S3 method for class 'permat'
summary(object, ...)
## S3 method for class 'summary.permat'
print(x, digits = 2, ...)
## S3 method for class 'permat'
plot(x, type = "bray", ylab, xlab, col, lty,
    lowess = TRUE, plot = TRUE, text = TRUE, ...)
## S3 method for class 'permat'
lines(x, type = "bray", ...)
## S3 method for class 'permat'
as.ts(x, type = "bray", ...)
## S3 method for class 'permat'
toCoda(x)
```

Arguments

`m`	A community data matrix with plots (samples) as rows and species (taxa) as columns.
`fixedmar`	character, stating which of the row/column sums should be preserved (`"none"`, `"rows"`, `"columns"`, `"both"`).

strata	Numeric vector or factor with length same as nrow(m) for grouping rows within strata for restricted permutations. Unique values or levels are used.
mtype	Matrix data type, either "count" for count data, or "prab" for presence-absence type incidence data.
times	Number of permuted matrices.
method	Character for method used for the swap algorithm ("swap", "tswap", "quasiswap", "backtrack") as described for function make.commsim. If mtype="count" the "quasiswap", "swap", "swsh" and "abuswap" methods are available (see details).
shuffle	Character, indicating whether individuals ("ind"), samples ("samp") or both ("both") should be shuffled, see details.
burnin	Number of null communities discarded before proper analysis in sequential ("swap", "tswap") methods.
thin	Number of discarded permuted matrices between two evaluations in sequential ("swap", "tswap") methods.
x, object	Object of class "permat"
digits	Number of digits used for rounding.
ylab, xlab, col, lty	graphical parameters for the plot method.
type	Character, type of plot to be displayed: "bray" for Bray-Curtis dissimilarities, "chisq" for Chi-squared values.
lowess, plot, text	Logical arguments for the plot method, whether a locally weighted regression curve should be drawn, the plot should be drawn, and statistic values should be printed on the plot.
...	Other arguments passed to simulate.nullmodel or methods.

Details

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

The shuffle argument only have effect if the mtype = "count" and permatfull function is used with "none", "rows", "columns" values of fixedmar. All other cases for count data are individual based randomisations. The "samp" and "both" options result fixed matrix fill. The "both" option means that individuals are shuffled among non zero cells ensuring that there are no cell with zeros as a result, then cell (zero and new valued cells) are shuffled.

The function permatswap is useful when with matrix fill (i.e. the proportion of empty cells) and row/columns sums should be kept constant. permatswap uses different kinds of swap algorithms, and row and columns sums are fixed in all cases. For presence-absence data, the swap and tswap

methods of make.commsim can be used. For count data, a special swap algorithm ('swapcount') is implemented that results in permuted matrices with fixed marginals and matrix fill at the same time.

The 'quasiswapcount' algorithm (method="quasiswap" and mtype="count") uses the same trick as Carsten Dormann's swap.web function in the package **bipartite**. First, a random matrix is generated by the r2dtable function retaining row and column sums. Then the original matrix fill is reconstructed by sequential steps to increase or decrease matrix fill in the random matrix. These steps are based on swapping 2x2 submatrices (see 'swapcount' algorithm for details) to maintain row and column totals. This algorithm generates independent matrices in each step, so burnin and thin arguments are not considered. This is the default method, because this is not sequential (as swapcount is) so independence of subsequent matrices does not have to be checked.

The swapcount algorithm (method="swap" and mtype="count") tries to find 2x2 submatrices (identified by 2 random row and 2 random column indices), that can be swapped in order to leave column and row totals and fill unchanged. First, the algorithm finds the largest value in the submatrix that can be swapped (d) and whether in diagonal or antidiagonal way. Submatrices that contain values larger than zero in either diagonal or antidiagonal position can be swapped. Swap means that the values in diagonal or antidiagonal positions are decreased by d, while remaining cells are increased by d. A swap is made only if fill doesn't change. This algorithm is sequential, subsequent matrices are not independent, because swaps modify little if the matrix is large. In these cases many burnin steps and thinning is needed to get independent random matrices. Although this algorithm is implemented in C, large burnin and thin values can slow it down considerably. WARNING: according to simulations, this algorithm seems to be biased and non random, thus its use should be avoided!

The algorithm "swsh" in the function permatswap is a hybrid algorithm. First, it makes binary quasiswaps to keep row and column incidences constant, then non-zero values are modified according to the shuffle argument (only "samp" and "both" are available in this case, because it is applied only on non-zero values). It also recognizes the fixedmar argument which cannot be "both" (**vegan** versions <= 2.0 had this algorithm with fixedmar = "none").

The algorithm "abuswap" produces two kinds of null models (based on fixedmar="columns" or fixedmar="rows") as described in Hardy (2008; randomization scheme 2x and 3x, respectively). These preserve column and row occurrences, and column or row sums at the same time. (Note that similar constraints can be achieved by the non sequential "swsh" algorithm with fixedmar argument set to "columns" or "rows", respectively.)

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

Unrestricted and restricted permutations: if strata is NULL, functions perform unrestricted permutations. Otherwise, it is used for restricted permutations. Each strata should contain at least 2 rows in order to perform randomization (in case of low row numbers, swap algorithms can be rather slow). If the design is not well balanced (i.e. same number of observations within each stratum), permuted matrices may be biased because same constraints are forced on submatrices of different dimensions. This often means, that the number of potential permutations will decrease with their dimensions. So the more constraints we put, the less randomness can be expected.

The plot method is useful for graphically testing for trend and independence of permuted matrices. This is especially important when using sequential algorithms ("swap", "tswap", "abuswap").

The as.ts method can be used to extract Bray-Curtis dissimilarities or Chi-squared values as time series. This can further used in testing independence (see Examples). The method toCoda is useful for accessing diagnostic tools available in the **coda** package.

Value

Functions permatfull and permatswap return an object of class "permat" containing the the function call (call), the original data matrix used for permutations (orig) and a list of permuted matrices with length times (perm).

The summary method returns various statistics as a list (including mean Bray-Curtis dissimilarities calculated pairwise among original and permuted matrices, Chi-square statistics, and check results of the constraints; see Examples). Note that when strata is used in the original call, summary calculation may take longer.

The plot creates a plot as a side effect.

The as.ts method returns an object of class "ts".

Author(s)

Péter Sólymos, <solymos@ualberta.ca> and Jari Oksanen

References

Original references for presence-absence algorithms are given on help page of make.commsim.

Hardy, O. J. (2008) Testing the spatial phylogenetic structure of local communities: statistical performances of different null models and test statistics on a locally neutral community. Journal of Ecology 96, 914–926.

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

See Also

For other functions to permute matrices: make.commsim, r2dtable, sample.

For the use of these permutation algorithms: oecosimu, adipart, hiersimu.

For time-series diagnostics: Box.test, lag.plot, tsdiag, ar, arima

For underlying low level implementation: commsim and nullmodel.

Examples

```
## A simple artificial community data matrix.
m <- matrix(c(
    1,3,2,0,3,1,
    0,2,1,0,2,1,
    0,0,1,2,0,3,
    0,0,0,1,4,3
    ), 4, 6, byrow=TRUE)
## Using the quasiswap algorithm to create a
## list of permuted matrices, where
## row/columns sums and matrix fill are preserved:
```

```
x1 <- permatswap(m, "quasiswap")
summary(x1)
## Unrestricted permutation retaining
## row/columns sums but not matrix fill:
x2 <- permatfull(m)
summary(x2)
## Unrestricted permutation of presence-absence type
## not retaining row/columns sums:
x3 <- permatfull(m, "none", mtype="prab")
x3$orig  ## note: original matrix is binarized!
summary(x3)
## Restricted permutation,
## check sums within strata:
x4 <- permatfull(m, strata=c(1,1,2,2))
summary(x4)

## NOTE: 'times' argument usually needs to be >= 99
## here much lower value is used for demonstration

## Not sequential algorithm
data(BCI)
a <- permatswap(BCI, "quasiswap", times=19)
## Sequential algorithm
b <- permatswap(BCI, "abuswap", fixedmar="col",
    burnin=0, thin=100, times=19)
opar <- par(mfrow=c(2,2))
plot(a, main="Not sequential")
plot(b, main="Sequential")
plot(a, "chisq")
plot(b, "chisq")
par(opar)
## Extract Bray-Curtis dissimilarities
## as time series
bc <- as.ts(b)
## Lag plot
lag.plot(bc)
## First order autoregressive model
mar <- arima(bc, c(1,0,0))
mar
## Ljung-Box test of residuals
Box.test(residuals(mar))
## Graphical diagnostics
tsdiag(mar)
```

`permustats` *Extract, Analyse and Display Permutation Results*

Description

The `permustats` function extracts permutation results of **vegan** functions. Its support functions can find quantiles and standardized effect sizes, plot densities and Q-Q plots.

Usage

```
permustats(x, ...)
## S3 method for class 'permustats'
summary(object, interval = 0.95, alternative, ...)
## S3 method for class 'permustats'
density(x, observed = TRUE, which, ...)
## S3 method for class 'permustats'
qqnorm(y, x, observed = TRUE, which, xlab, ylab, ...)
## S3 method for class 'permustats'
boxplot(x, scale = FALSE, names, ...)
## S3 method for class 'permustats'
pairs(x, ...)
```

Arguments

`object`, `x`, `y`	The object to be handled. Usually this is `x`, but in `qqnorm` the permutation values are displayed on y-axis, and `x` can be used for user-supplied quantiles.
`interval`	numeric; the coverage interval reported.
`alternative`	A character string specifying the limits used for the `interval` and the direction of the test when evaluating the p-values. Must be one of `"two.sided"` (both upper and lower limit), `"greater"` (upper limit), `"less"` (lower limit). Usually `alternative` is given in the result object, but it can be specified with this argument.
`observed`	Add observed statistic among permutations.
`which`	`which` (integer) of permutation statistics is displayed.
`xlab`, `ylab`	Axis labels to replace defaults.
`scale`	Use standardized effect size (SES).
`names`	Names of boxes (default: names of statistics).
`...`	Other arguments passed to the function. In `density` these are passed to `density.default`, and in `boxplot` to `boxplot.default`.

Details

The `permustats` function extracts permutation results and observed statistics from several **vegan** functions that perform permutations or simulations.

The `summary` method of `permustats` estimates the standardized effect sizes (SES) as the difference of observed statistic and mean of permutations divided by the standard deviation of permutations (also known as z-values). It also prints the the mean, median, and limits which contain `interval` percent of permuted values. With the default (`interval = 0.95`), for two-sided test these are (2.5%, 97.5%) and for one-sided tests either 5% or 95% quantile and the p-value depending on the test direction. The mean, quantiles and z values are evaluated from permuted values without observed statistic, but the p-value is evaluated with the observed statistic. The intervals and the p-value are evaluated with the same test direction as in the original test, but this can be changed with argument `alternative`. Several `permustats` objects can be combined with `c` function. The `c` function checks that statistics are equal, but performs no other sanity tests.

The results can be displayed with conventional graphics or as **ggplot2** graphics using `autoplot` function in **ggvegan** package.

The `density` prepares kernel density estimates of permuted values; these can be displayed with `plot`. Only one statistic can be displayed: use `which` to select the displayed statistic.

The `qqnorm` method displays Q-Q plots of permutations, optionally together with the observed value (default) which is shown as horizontal line in plots. `qqnorm` plots permutation values against standard Normal variate, but users can supply other reference axis with argument `x`. The permutations are standardized without the observed statistic, similarly as in `summary`. Only one statistic can be displayed: use `which` to select the displayed statistic.

Function `boxplot` draws the box-and-whiskers plots of effect size, or the difference of permutations and observed statistic. If `scale = TRUE`, permutations are standardized to unit standard deviation, and the plot will show the standardized effect sizes.

Function `pairs` plots permutation values of statistics against each other. The function passes extra arguments to `pairs`.

The `permustats` can extract permutation statistics from the results of `adonis2`, `anosim`, `anova.cca`, `mantel`, `mantel.partial`, `mrpp`, `oecosimu`, `ordiareatest`, `permutest.cca`, `protest`, and `permutest.betadisper`.

Value

The `permustats` function returns an object of class `"permustats"`. This is a list of items `"statistic"` for observed statistics, `permutations` which contains permuted values, and `alternative` which contains text defining the character of the test (`"two.sided"`, `"less"` or `"greater"`). The `qqnorm` and `density` methods return their standard result objects.

Author(s)

Jari Oksanen with contributions from Gavin L. Simpson (`permustats.permutest.betadisper` method and related modifications to `summary.permustats` and the `print` method) and Eduard Szöcs (`permustats.anova.cca`).

See Also

`density`, `qqnorm`, `boxplot`. Package **ggvegan** may have (depending on version) alternative functions to display `permustats` results.

Examples

```
data(dune, dune.env)
mod <- adonis2(dune ~ Management + A1, data = dune.env)
## use permustats
perm <- permustats(mod)
summary(perm)
boxplot(perm, scale=TRUE, lty=1, pch=16, cex=0.6, col="hotpink", ylab="SES")
abline(h=0, col="skyblue")
## example of multiple types of statistic
mod <- with(dune.env, betadisper(vegdist(dune), Management))
pmod <- permutest(mod, nperm = 99, pairwise = TRUE)
perm <- permustats(pmod)
summary(perm, interval = 0.90)
```

`permutations` *Permutation tests in Vegan*

Description

From version 2.2-0, **vegan** has significantly improved access to restricted permutations which brings it into line with those offered by Canoco. The permutation designs are modelled after the permutation schemes of Canoco 3.1 (ter Braak, 1990).

vegan currently provides for the following features within permutation tests:

1. Free permutation of *DATA*, also known as randomisation,
2. Free permutation of *DATA* within the levels of a grouping variable,
3. Restricted permutations for line transects or time series,
4. Permutation of groups of samples whilst retaining the within-group ordering,
5. Restricted permutations for spatial grids,
6. Blocking, samples are never permuted *between* blocks, and
7. Split-plot designs, with permutation of whole plots, split plots, or both.

Above, we use *DATA* to mean either the observed data themselves or some function of the data, for example the residuals of an ordination model in the presence of covariables.

These capabilities are provided by functions from the **permute** package. The user can request a particular type of permutation by supplying the `permutations` argument of a function with an object returned by `how`, which defines how samples should be permuted. Alternatively, the user can simply specify the required number of permutations and a simple randomisation procedure will be performed. Finally, the user can supply a matrix of permutations (with number of rows equal to the number of permutations and number of columns equal to the number of observations in the data) and **vegan** will use these permutations instead of generating new permutations.

The majority of functions in **vegan** allow for the full range of possibilities outlined above. Exceptions include `kendall.post` and `kendall.global`.

The Null hypothesis for the first two types of permutation test listed above assumes free exchangeability of *DATA* (within the levels of the grouping variable, if specified). Dependence between observations, such as that which arises due to spatial or temporal autocorrelation, or more-complicated experimental designs, such as split-plot designs, violates this fundamental assumption of the test and requires more complex restricted permutation test designs. It is these designs that are available via the **permute** package and to which **vegan** provides access from version 2.2-0 onwards.

Unless otherwise stated in the help pages for specific functions, permutation tests in **vegan** all follow the same format/structure:

1. An appropriate test statistic is chosen. Which statistic is chosen should be described on the help pages for individual functions.
2. The value of the test statistic is evaluate for the observed data and analysis/model and recorded. Denote this value x_0.
3. The *DATA* are randomly permuted according to one of the above schemes, and the value of the test statistic for this permutation is evaluated and recorded.

4. Step 3 is repeated a total of n times, where n is the number of permutations requested. Denote these values as x_i, where $i = 1, ..., n$
5. Count the number of values of the test statistic, x_i, in the Null distribution that are as extreme as test statistic for the observed data x_0. Denote this count as N.

   We use the phrase *as extreme* to include cases where a two-sided test is performed and large negative values of the test statistic should be considered.
6. The permutation p-value is computed as

$$p = \frac{N+1}{n+1}$$

The above description illustrates why the default number of permutations specified in **vegan** functions takes values of 199 or 999 for example. Pretty *p* values are achieved because the $+1$ in the denominator results in division by 200 or 1000, for the 199 or 999 random permutations used in the test.

The simple intuition behind the presence of $+1$ in the numerator and denominator is that these represent the inclusion of the observed value of the statistic in the Null distribution (e.g. Manly 2006). Phipson & Smyth (2010) present a more compelling explanation for the inclusion of $+1$ in the numerator and denominator of the *p* value calculation.

Fisher (1935) had in mind that a permutation test would involve enumeration of all possible permutations of the data yielding an exact test. However, doing this complete enumeration may not be feasible in practice owing to the potentially vast number of arrangements of the data, even in modestly-sized data sets with free permutation of samples. As a result we evaluate the *p* value as the tail probability of the Null distribution of the test statistic directly from the random sample of possible permutations. Phipson & Smyth (2010) show that the naive calculation of the permutation *p* value is

$$p = \frac{N}{n}$$

which leads to an invalid test with incorrect type I error rate. They go on to show that by replacing the unknown tail probability (the *p* value) of the Null distribution with the biased estimator

$$p = \frac{N+1}{n+1}$$

that the positive bias induced is of just the right size to account for the uncertainty in the estimation of the tail probability from the set of randomly sampled permutations to yield a test with the correct type I error rate.

The estimator described above is correct for the situation where permutations of the data are samples randomly *without* replacement. This is not strictly what happens in **vegan** because permutations are drawn pseudo-randomly independent of one another. Note that the actual chance of this happening is practice is small but the functions in **permute** do not guarantee to generate a unique set of permutations unless complete enumeration of permutations is requested. This is not feasible for all but the smallest of data sets or restrictive of permutation designs, but in such cases the chance of drawing a set of permutations with repeats is lessened as the sample size, and thence the size of set of all possible permutations, increases.

Under the situation of sampling permutations with replacement then, the tail probability p calculated from the biased estimator described above is somewhat **conservative**, being too large by an amount that depends on the number of possible values that the test statistic can take under permutation of the data (Phipson & Smyth, 2010). This represents a slight loss of statistical power for the conservative p value calculation used here. However, unless sample sizes are small and the the permutation design such that the set of values that the test statistic can take is also small, this loss of power is unlikely to be critical.

The minimum achievable p-value is

$$p_{\min} = \frac{1}{n+1}$$

and hence depends on the number of permutations evaluated. However, one cannot simply increase the number of permutations (n) to achieve a potentially lower p-value unless the number of observations available permits such a number of permutations. This is unlikely to be a problem for all but the smallest data sets when free permutation (randomisation) is valid, but in restricted permutation designs with a low number of observations, there may not be as many unique permutations of the data as you might desire to reach the required level of significance.

It is currently the responsibility of the user to determine the total number of possible permutations for their *DATA*. The number of possible permutations allowed under the specified design can be calculated using `numPerms` from the **permute** package. Heuristics employed within the `shuffleSet` function used by **vegan** can be triggered to generate the entire set of permutations instead of a random set. The settings controlling the triggering of the complete enumeration step are contained within a permutation design created using `link[permute]{how}` and can be set by the user. See `how` for details.

Limits on the total number of permutations of *DATA* are more severe in temporally or spatially ordered data or experimental designs with low replication. For example, a time series of $n = 100$ observations has just 100 possible permutations **including** the observed ordering.

In situations where only a low number of permutations is possible due to the nature of *DATA* or the experimental design, enumeration of all permutations becomes important and achievable computationally.

Above, we have provided only a brief overview of the capabilities of **vegan** and **permute**. To get the best out of the new functionality and for details on how to set up permutation designs using `how`, consult the vignette *Restricted permutations; using the permute package* supplied with **permute** and accessible via `vignette("permutations", package = "permute")`.

Random Number Generation

The permutations are based on the random number generator provided by R. This may change in R releases and change the permutations and **vegan** test results. One such change was in R release 3.6.0. The new version is clearly better for permutation tests and you should use it. However, if you need to reproduce old results, you can set the R random number generator to a previous version with `RNGversion`.

Author(s)

Gavin L. Simpson

References

Manly, B. F. J. (2006). *Randomization, Bootstrap and Monte Carlo Methods in Biology*, Third Edition. Chapman and Hall/CRC.

Phipson, B., & Smyth, G. K. (2010). Permutation P-values should never be zero: calculating exact P-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology*, **9**, Article 39. DOI: 10.2202/1544-6115.1585

ter Braak, C. J. F. (1990). *Update notes: CANOCO version 3.1*. Wageningen: Agricultural Mathematics Group. (UR).

See also:

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and their Application*. Cambridge University Press.

See Also

`permutest` for the main interface in **vegan**. See also `how` for details on permutation design specification, `shuffleSet` for the code used to generate a set of permutations, `numPerms` for a function to return the size of the set of possible permutations under the current design.

`permutest.betadisper` *Permutation test of multivariate homogeneity of groups dispersions (variances)*

Description

Implements a permutation-based test of multivariate homogeneity of group dispersions (variances) for the results of a call to `betadisper`.

Usage

```
## S3 method for class 'betadisper'
permutest(x, pairwise = FALSE,
          permutations = 999,
          parallel = getOption("mc.cores"),
          ...)
```

Arguments

`x`	an object of class `"betadisper"`, the result of a call to `betadisper`.
`pairwise`	logical; perform pairwise comparisons of group means?
`permutations`	a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
`parallel`	Number of parallel processes or a predefined socket cluster. With `parallel = 1` uses ordinary, non-parallel processing.
`...`	Arguments passed to other methods.

Details

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

Pairwise comparisons of group mean dispersions can be performed by setting argument `pairwise` to `TRUE`. A classical t test is performed on the pairwise group dispersions. This is combined with a permutation test based on the t statistic calculated on pairwise group dispersions. An alternative to the classical comparison of group dispersions, is to calculate Tukey's Honest Significant Differences between groups, via `TukeyHSD.betadisper`.

Value

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

`tab`	the ANOVA table which is an object inheriting from class `"data.frame"`.
`pairwise`	a list with components `observed` and `permuted` containing the observed and permuted p-values for pairwise comparisons of group mean distances (dispersions or variances).
`groups`	character; the levels of the grouping factor.
`control`	a list, the result of a call to `how`.

Author(s)

Gavin L. Simpson

References

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

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

See Also

For the main fitting function see `betadisper`. For an alternative approach to determining which groups are more variable, see `TukeyHSD.betadisper`.

Examples

```
data(varespec)

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

## First 16 sites grazed, remaining 8 sites ungrazed
```

```
groups <- factor(c(rep(1,16), rep(2,8)), labels = c("grazed","ungrazed"))

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

## Perform test
anova(mod)

## Permutation test for F
pmod <- permutest(mod, permutations = 99, pairwise = TRUE)

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

---

plot.cca *Plot or Extract Results of Constrained Correspondence Analysis or Redundancy Analysis*

Description

Functions to plot or extract results of constrained correspondence analysis (cca), redundancy analysis (rda), distance-based redundancy analysis (dbrda) 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 = "species", type, xlim, ylim, const,
     correlation = FALSE, hill = FALSE, optimize = FALSE, arrows = FALSE,
     spe.par = list(), sit.par = list(), con.par = list(), bip.par = list(),
     cen.par = list(), reg.par = list(), ...)
## S3 method for class 'cca'
text(x, display = "sites", labels, choices = c(1, 2),
     scaling = "species", arrow.mul, head.arrow = 0.05, select, const,
     correlation = FALSE, hill = FALSE, ...)
## S3 method for class 'cca'
points(x, display = "sites", choices = c(1, 2),
     scaling = "species", arrow.mul, head.arrow = 0.05, select, const,
     correlation = FALSE, hill = FALSE, ...)
## S3 method for class 'cca'
scores(x, choices = c(1,2), display = "all",
     scaling = "species", hill = FALSE, tidy = FALSE, droplist = TRUE,
     ...)
## S3 method for class 'rda'
scores(x, choices = c(1,2), display = "all",
```

```
     scaling = "species", const, correlation = FALSE, tidy = FALSE,
     droplist = TRUE, ...)
## S3 method for class 'cca'
summary(object, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'cca'
labels(object, display, ...)
```

Arguments

`x`, `object` A `cca` result object.

`choices` Axes shown.

`display` Scores shown. These must include some of the alternatives `"species"` or `"sp"` for species scores, `sites` or `"wa"` for site scores, `"lc"` for linear constraints or LC scores, or `"bp"` for biplot arrows or `"cn"` for centroids of factor constraints instead of an arrow, and `"reg"` for regression coefficients (a.k.a. canonical coefficients). The alternative `"all"` selects all available scores.

`scaling` Scaling for species and site scores. Either species (`2`) or site (`1`) scores are scaled by eigenvalues, and the other set of scores is left unscaled, or with `3` both are scaled symmetrically by square root of eigenvalues. Corresponding negative values can be used in `cca` to additionally multiply results with $\sqrt{(1/(1-\lambda))}$. This scaling is know as Hill scaling (although it has nothing to do with Hill's rescaling of `decorana`). With corresponding negative values in `rda`, species scores are divided by standard deviation of each species and multiplied with an equalizing constant. Unscaled raw scores stored in the result can be accessed with `scaling = 0`.

The type of scores can also be specified as one of `"none"`, `"sites"`, `"species"`, or `"symmetric"`, which correspond to the values `0`, `1`, `2`, and `3` respectively. Arguments `correlation` and `hill` in `scores.rda` and `scores.cca` respectively can be used in combination with these character descriptions to get the corresponding negative value.

`correlation`, `hill`
logical; if `scaling` is a character description of the scaling type, `correlation` or `hill` are used to select the corresponding negative scaling type; either correlation-like scores or Hill's scaling for PCA/RDA and CA/CCA respectively. See argument `scaling` for details.

`optimize` Optimize locations of text to reduce overlap and plot point in the actual locations of the scores. Uses `ordipointlabel`.

`arrows` Draw arrows from the origin. This will always be `TRUE` for biplot and regression scores in constrained ordination (`cca` etc.). Setting this `TRUE` will draw arrows for any type of scores. This allows, e.g, using biplot arrows for species. The arrow head will be at the value of scores, and possible text is moved outwards or somewhere around the arrowhead with argument `optimize = TRUE` (possibly over the arrow).

`spe.par`, `sit.par`, `con.par`, `bip.par`, `cen.par`, `reg.par`
Lists of graphical parameters for species, sites, constraints (lc scores), biplot and text, centroids and regression. These take precedence over globally set parameters and defaults.

tidy	Return scores that are compatible with **ggvegan**: all scores are in a single `data.frame`, score type is identified by factor variable `score`, the names by variable `label`, and weights (in CCA) are in variable `weight`. The possible values of `score` are `species`, `sites` (for WA scores), `constraints` (LC scores for sites calculated directly from the constraining variables), `biplot` (for biplot arrows), `centroids` (for levels of factor variables), `factorbiplot` (biplot arrows that model centroids), `regression` (for regression coefficients to find LC scores from constraints). These scores cannot be used with conventional `plot`, but they are directly suitable to be used with the **ggvegan** package.
type	Type of plot: partial match to `text` for text labels, `points` for points, and `none` for setting frames only. If omitted, `text` is selected for smaller data sets, and `points` for larger.
xlim, ylim	the x and y limits (min,max) of the plot.
labels	Optional text to be used for selected items instead of row names. If you use this, it is good to check the default labels and their order using `labels` command. If `select` is given, give `labels` only to the selected items.
arrow.mul	Factor to expand arrows in the graph. Arrows will be scaled automatically to fit the graph if this is missing.
head.arrow	Default length of arrow heads.
select	Items to be displayed. This can either be a logical vector which is `TRUE` for displayed items or a vector of indices of displayed items.
const	General scaling constant to `rda` scores. The default is to use a constant that gives biplot scores, that is, scores that approximate original data (see `vignette` on 'Design Decisions' with `browseVignettes("vegan")` for details and discussion). If `const` is a vector of two items, the first is used for species, and the second item for site scores.
droplist	Return a matrix instead of a named list when only one kind of scores were requested.
digits	Number of digits in output.
...	Parameters passed to graphical functions. These will be applied to all score types, but will be superseded by score type parameters list (except `type = "none"` which will only draw the frame).

Details

Same `plot` function will be used for `cca`, `rda`, `dbrda` and `capscale`. This produces a quick, standard plot with current `scaling`.

The `plot` function sets colours (`col`), plotting characters (`pch`) and character sizes (`cex`) to default values for each score type. The defaults can be changed with global parameters ("dot arguments") applied to all score types, or a list of parameters for a specified score type (`spe.par`, `sit.par` etc.) which take precedence over global parameters and defaults. This allows full control of graphics. The scores are plotted with `text.ordiplot` and `points.ordiplot` and accept parameters of these functions. In addition to standard graphical parameters, text can be plotted over non-transparent label with arbument `bg = <colour>`, and location of text can be optimized to avoid over-writing with argument `optimize = TRUE`, and argument `arrows = TRUE` to draw arrows pointing to the ordination scores.

the plot function returns (invisible) ordiplot object. You can save this object and use it to construct your plot with ordiplot functions points and text. These functions can be used in pipe (|>) which allows incremental building of plots with full control of graphical parameters for each score type. With pipe it is best to first create an empty plot with plot(<cca-result>, type = "n") and then add elements with points, text of ordiplot or ordilabel. Within pipe, the first argument should be a quoted score type, and then the grapcical parameters. The full object may contain scores with names 'species', 'sites', 'constraints', 'biplot', 'regression', 'centroids' (some of these may be missing depending on your model and are only available if given in display). The first plot will set the dimensions of graph, and if you do not use some score type there may be empty white space. In addition to ordiplot text and points, you can also use ordilabel and ordipointlabel in a pipe. Unlike in basic plot, there are no defaults for score types, but all graphical parameters must be set in the command in pipe. On the other hand, there may be more flexibility in these settings than in plot arguments, in particular in ordilabel and ordipointlabel.

Environmental variables receive a special treatment. With display="bp", arrows will be drawn. These are labelled with text and unlabelled with points. The arrows have basically unit scaling, but if sites were scaled (scaling "sites" or "symmetric"), the scores of requested axes are adjusted to the plotting area. With scaling = "species" or scaling = "none", the arrows will be consistent with vectors fitted to linear combination scores (display = "lc" in function envfit), but with other scaling alternatives they will differ. The basic plot function uses a simple heuristics for adjusting the unit-length arrows to the current plot area, but the user can give the expansion factor in arrow.mul. With display="cn" the centroids of levels of factor variables are displayed. With this option continuous variables still are presented as arrows and ordered factors as arrows and centroids. With display = "reg" arrows will be drawn for regression coefficients (a.k.a. canonical coefficients) of constraints and conditions. Biplot arrows can be interpreted individually, but regression coefficients must be interpreted all together: the LC score for each site is the sum of regressions displayed by arrows. The partialled out conditions are zero and not shown in biplot arrows, but they are shown for regressions, and show the effect that must be partialled out to get the LC scores. The biplot arrows are more standard and more easily interpreted, and regression arrows should be used only if you know that you need them.

The ordination object has text and points methods that can be used to add items to an existing plot from the ordination result directly. These should be used with extreme care, because you must set scaling and other graphical parameters exactly similarly as in the original plot command. It is best to avoid using these historic functions and instead configure plot command or use pipe.

Palmer (1993) suggested using linear constraints ("LC scores") in ordination diagrams, because these gave better results in simulations and site scores ("WA scores") are a step from constrained to unconstrained analysis. However, McCune (1997) showed that noisy environmental variables (and all environmental measurements are noisy) destroy "LC scores" whereas "WA scores" were little affected. Therefore the plot function uses site scores ("WA scores") as the default. This is consistent with the usage in statistics and other functions in R (lda, cancor).

Value

The plot function returns invisibly a plotting structure which can be used by function identify.ordiplot to identify the points or other functions in the ordiplot family or in a pipe to add new graphicael elements with ordiplot text and points or with ordilabel and ordipointlabel.

Author(s)

Jari Oksanen

See Also

The function builds upon ordiplot and its text and points functions. See these to find new graphical parameters such as arrows (for drawing arrows), bg (for writing text on non-transparent label) and optimize (to move text labels of points to avoid overwriting).

Examples

```
data(dune, dune.env)
mod <- cca(dune ~ Moisture + Management, dune.env)

## default and modified plot
plot(mod, scaling="sites")
plot(mod, scaling="sites", type = "text",
  sit.par = list(type = "points", pch=21, col="red", bg="yellow", cex=1.2),
  spe.par = list(col="blue", cex=0.8, optimize = TRUE),
  cen.par = list(bg="white"))

## same with pipe
plot(mod, type="n", scaling="sites") |>
  points("sites", pch=21, col="red", bg = "yellow", cex=1.2) |>
  text("species", col="blue", cex=0.8) |>
  text("biplot") |>
  text("centroids", bg="white")

## LC scores & factors mean much overplotting: try optimize=TRUE
plot(mod, display = c("lc","sp","cn"), optimize = TRUE)

## catch the invisible result and use ordiplot support - the example
## will make a biplot with arrows for species and correlation scaling
pca <- rda(dune)
pl <- plot(pca, type="n", scaling="sites", correlation=TRUE)
with(dune.env, points(pl, "site", pch=21, col=1, bg=Management))
text(pl, "sp", arrow=TRUE, length=0.05, col=4, cex=0.6, xpd=TRUE)
with(dune.env, legend("bottomleft", levels(Management), pch=21,
   pt.bg=1:4, bty="n"))

## Pipe
plot(pca, type="n", scaling="sites", correlation=TRUE) |>
    points("sites", pch=21, col = 1, cex=1.5, bg = dune.env$Management) |>
    text("species", col = "blue", arrows = TRUE, xpd = TRUE, font = 3)

## Scaling can be numeric or more user-friendly names
## e.g. Hill's scaling for (C)CA
scrs <- scores(mod, scaling = "sites", hill = TRUE)
## or correlation-based scores in PCA/RDA
scrs <- scores(rda(dune ~ A1 + Moisture + Management, dune.env),
               scaling = "sites", correlation = TRUE)
```

prc *Principal Response Curves for Treatments with Repeated Observations*

Description

Principal Response Curves (PRC) are a special case of Redundancy Analysis (rda) for multivariate responses in repeated observation design. They were originally suggested for ecological communities. They should be easier to interpret than traditional constrained ordination. They can also be used to study how the effects of a factor A depend on the levels of a factor B, that is A + A:B, in a multivariate response experiment.

Usage

```
prc(response, treatment, time, ...)
## S3 method for class 'prc'
summary(object, axis = 1, scaling = "sites", const,
        digits = 4, correlation = FALSE, ...)
## S3 method for class 'prc'
plot(x, species = TRUE, select, scaling = "symmetric",
     axis = 1, correlation = FALSE, const, type = "l", xlab, ylab, ylim,
     lty = 1:5, col = 1:6, pch, legpos, cex = 0.8, ...)
```

Arguments

response	Multivariate response data. Typically these are community (species) data. If the data are counts, they probably should be log transformed prior to the analysis.
treatment	A factor for treatments.
time	An unordered factor defining the observations times in the repeated design.
object, x	An prc result object.
axis	Axis shown (only one axis can be selected).
scaling	Scaling of species scores, identical to the scaling in scores.rda. The type of scores can also be specified as one of "none", "sites", "species", or "symmetric", which correspond to the values 0, 1, 2, and 3 respectively. Argument correlation can be used in combination with these character descriptions to get the corresponding negative value.
const	General scaling constant for species scores (see scores.rda for details). Lower values will reduce the range of species scores, but will not influence the regression coefficients.
digits	Number of significant digits displayed.
correlation	logical; if scaling is a character description of the scaling type, correlation can be used to select correlation-like scores for PCA. See argument scaling for details.
species	Display species scores.

select	Vector to select displayed species. This can be a vector of indices or a logical vector which is TRUE for the selected species
type	Type of plot: "l" for lines, "p" for points or "b" for both.
xlab, ylab	Text to replace default axis labels.
ylim	Limits for the vertical axis.
lty, col, pch	Line type, colour and plotting characters (defaults supplied).
legpos	The position of the legend. A guess is made if this is not supplied, and NA will suppress legend.
cex	Character expansion for symbols and species labels.
...	Other parameters passed to functions.

Details

PRC is a special case of rda with a single factor for treatment and a single factor for time points in repeated observations. In **vegan**, the corresponding rda model is defined as rda(response ~ treatment * time + Condition(time)). Since the time appears twice in the model formula, its main effects will be aliased, and only the main effect of treatment and interaction terms are available, and will be used in PRC. Instead of usual multivariate ordination diagrams, PRC uses canonical (regression) coefficients and species scores for a single axis. All that the current functions do is to provide a special summary and plot methods that display the rda results in the PRC fashion. The current version only works with default contrasts (contr.treatment) in which the coefficients are contrasts against the first level, and the levels must be arranged so that the first level is the control (or a baseline). If necessary, you must change the baseline level with function relevel.

Function summary prints the species scores and the coefficients. Function plot plots coefficients against time using matplot, and has similar defaults. The graph (and PRC) is meaningful only if the first treatment level is the control, as the results are contrasts to the first level when unordered factors are used. The plot also displays species scores on the right vertical axis using function linestack. Typically the number of species is so high that not all can be displayed with the default settings, but users can reduce character size or padding (air) in linestack, or select only a subset of the species. A legend will be displayed unless suppressed with legpos = NA, and the functions tries to guess where to put the legend if legpos is not supplied.

Value

The function is a special case of rda and returns its result object (see cca.object). However, a special summary and plot methods display returns differently than in rda.

Warning

The first level of treatment must be the control: use function relevel to guarantee the correct reference level. The current version will ignore user setting of contrasts and always use treatment contrasts (contr.treatment). The time must be an unordered factor.

Author(s)

Jari Oksanen and Cajo ter Braak

References

van den Brink, P.J. & ter Braak, C.J.F. (1999). Principal response curves: Analysis of time-dependent multivariate responses of biological community to stress. Environmental Toxicology and Chemistry, 18, 138–148.

See Also

rda, anova.cca.

Examples

```
## Chlorpyrifos experiment and experimental design: Pesticide
## treatment in ditches (replicated) and followed over from 4 weeks
## before to 24 weeks after exposure
data(pyrifos)
week <- gl(11, 12, labels=c(-4, -1, 0.1, 1, 2, 4, 8, 12, 15, 19, 24))
dose <- factor(rep(c(0.1, 0, 0, 0.9, 0, 44, 6, 0.1, 44, 0.9, 0, 6), 11))
ditch <- gl(12, 1, length=132)

## IGNORE_RDIFF_BEGIN
## PRC
mod <- prc(pyrifos, dose, week)
mod            # RDA
summary(mod)   # PRC
logabu <- colSums(pyrifos)
plot(mod, select = logabu > 100)
## IGNORE_RDIFF_END
## Ditches are randomized, we have a time series, and are only
## interested in the first axis
ctrl <- how(plots = Plots(strata = ditch,type = "free"),
    within = Within(type = "series"), nperm = 99)
anova(mod, permutations = ctrl, first=TRUE)
```

---

predict.cca *Prediction Tools for [Constrained] Ordination (CCA, RDA, DCA, CA, PCA)*

Description

Function predict can be used to find site and species scores or estimates of the response data with new data sets, Function calibrate estimates values of constraints with new data set. Functions fitted and residuals return estimates of response data.

Usage

```
## S3 method for class 'cca'
fitted(object, model = c("CCA", "CA", "pCCA"),
    type =  c("response", "working"), ...)
## S3 method for class 'capscale'
```

```
fitted(object, model = c("CCA", "CA", "pCCA", "Imaginary"),
    type = c("response", "working"), ...)
## S3 method for class 'cca'
residuals(object, ...)
## S3 method for class 'cca'
predict(object, newdata, type = c("response", "wa", "sp", "lc", "working"),
        rank = "full", model = c("CCA", "CA"), scaling = "none",
        hill = FALSE, ...)
## S3 method for class 'rda'
predict(object, newdata, type = c("response", "wa", "sp", "lc", "working"),
        rank = "full", model = c("CCA", "CA"), scaling = "none",
        correlation = FALSE, const, ...)
## S3 method for class 'dbrda'
predict(object, newdata, type = c("response", "lc", "wa", "working"),
        rank = "full", model = c("CCA", "CA"), scaling = "none", const, ...)
## S3 method for class 'cca'
calibrate(object, newdata, rank = "full", ...)
## S3 method for class 'cca'
coef(object, norm = FALSE, ...)
## S3 method for class 'decorana'
predict(object, newdata, type = c("response", "sites", "species"),
    rank = 4, ...)
```

Arguments

`object` A result object from `cca`, `rda`, `dbrda`, `capscale` or `decorana`.

`model` Show constrained (`"CCA"`), unconstrained (`"CA"`) or conditioned "partial" (`"pCCA"`) results. For `fitted` method of `capscale` this can also be `"Imaginary"` for imaginary components with negative eigenvalues

`newdata` New data frame to be used in prediction or in calibration. Usually this a new community data frame, but with `type = "lc"` and for constrained component with `type = "response"` and `type = "working"` it must be a data frame of constraints. The `newdata` must have the same number of rows as the original community data for a `cca` result with `type = "response"` or `type = "working"`. If the original model had row or column names, then new data must contain rows or columns with the same names (row names for species scores, column names for `"wa"` scores and constraint names of `"lc"` scores). In other cases the rows or columns must match directly. The argument is not implemented for `"wa"` scores in `dbrda`.

`type` The type of prediction, fitted values or residuals: `"response"` scales results so that the same ordination gives the same results, and `"working"` gives the values used internally, that is after Chi-square standardization in `cca` and scaling and centring in `rda`. In `capscale` and `dbrda` the `"response"` gives the dissimilarities, and `"working"` the internal data structure analysed in the ordination. Alternative `"wa"` gives 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	logical, character, or numeric; Scaling or predicted scores with the same meaning as in cca, rda, dbrda, and capscale. See scores.cca for further details on acceptable values.
correlation, hill	logical; correlation-like scores or Hill's scaling as appropriate for RDA and CCA respectively. See scores.cca for additional details.
const	Constant multiplier for RDA scores. This will be used only when scaling is not FALSE, and the default value will give similar scaling as in scores.rda.
norm	Coefficients for variables that are centred and scaled to unit norm.
...	Other parameters to the functions.

Details

Function fitted gives the approximation of the original data matrix or dissimilarities from the ordination result either in the scale of the response or as scaled internally by the function. Function residuals gives the approximation of the original data from the unconstrained ordination. With argument type = "response" the fitted.cca and residuals.cca function both give the same marginal totals as the original data matrix, and fitted and residuals do not add up to the original data. Functions fitted and residuals for dbrda and capscale give the dissimilarities with type = "response", but these are not additive. However, the "working" scores are additive for capscale (but not for dbrda). The fitted and residuals for capscale and dbrda will include the additive constant if that was requested in the function call. All variants of fitted and residuals 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 or dissimilarities (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 for cca or rda. The function can be used with new data, and it can be used to add new species or site scores to existing ordinations. The function returns (weighted) orthonormal scores by default, and you must specify explicit scaling to add those scores to ordination diagrams. With type = "wa" the function finds the site scores from species scores. In that case, the new data can contain new sites, but species must match in the original and new data. With type="sp" the function finds species scores from site constraints (linear combination scores). In that case the new data can contain new species, but sites must match in the original and new data. With type = "lc" the function finds the linear combination scores for sites from environmental data. In that case the new data frame must contain all constraining and conditioning environmental variables of the model formula. With type = "response" or type = "working" the new data must contain environmental variables if constrained component is desired, and community data matrix if residual or unconstrained component is desired. With these types, the function uses newdata to find new "lc" (constrained) or "wa" scores (unconstrained) and then finds the response or working data from these new row scores and species scores. The original site (row) and species (column) weights are used for type = "response" and type = "working" in correspondence analysis (cca) and therefore the number of rows must match in the original data and newdata.

If a completely new data frame is created, extreme care is needed defining variables similarly as in the original model, in particular with (ordered) factors. If ordination was performed with the

formula interface, the newdata can be a data frame or matrix, but extreme care is needed that the columns match in the original and newdata.

Function calibrate.cca finds estimates of constraints from community ordination or "wa" scores from cca, rda and capscale. This is often known as calibration, bioindication or environmental reconstruction, and it is equivalent to performing Weighted Averaging (see wascores). As a Weighted Averaging method it uses deshrinking where the sum of weighted prediction errors is zero. 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 or dbrda 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, vectors or dissimilarities as is appropriate.

Author(s)

Jari Oksanen.

References

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

See Also

cca, rda, dbrda, capscale, decorana, goodness.cca.

Examples

```
data(dune, 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)
```

```
## IGNORE_RDIFF_BEGIN
predict(mod, type="sp", newdata=dune[, freq==1], scaling="species")
# 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, dune.env)
head(pred <- calibrate(mod))
## For single variable similar to weighted averaging calibration, but
## different deshrinking
head(wascores(wascores(dune.env$A1, dune, expand=TRUE), t(dune), expand=TRUE))
## IGNORE_RDIFF_END
with(dune.env, plot(A1, pred - A1, ylab="Prediction Error"))
abline(h=0)
```

---

procrustes *Procrustes Rotation of Two Configurations and PROTEST*

Description

Function `procrustes` rotates a configuration to maximum similarity with another configuration. Function `protest` tests the non-randomness (significance) between two configurations.

Usage

```
procrustes(X, Y, scale = TRUE, symmetric = FALSE, scores = "sites", ...)
## S3 method for class 'procrustes'
summary(object, digits = getOption("digits"), ...)
## S3 method for class 'procrustes'
plot(x, kind=1, choices=c(1,2), to.target = TRUE,
    type = "p", xlab, ylab, main, ar.col = "blue", length=0.05,
    cex = 0.7, ...)
## S3 method for class 'procrustes'
points(x, display = c("target", "rotated"),
    choices = c(1,2), truemean = FALSE, ...)
## S3 method for class 'procrustes'
text(x, display = c("target", "rotated"),
    choices = c(1,2), labels, truemean = FALSE, ...)
## S3 method for class 'procrustes'
lines(x, type = c("segments", "arrows"),
    choices = c(1, 2), truemean = FALSE, ...)
## S3 method for class 'procrustes'
residuals(object, ...)
## S3 method for class 'procrustes'
fitted(object, truemean = TRUE, ...)
## S3 method for class 'procrustes'
predict(object, newdata, truemean = TRUE, ...)
protest(X, Y, scores = "sites", permutations = how(nperm = 999), ...)
```

Arguments

X	Target matrix
Y	Matrix to be rotated.
scale	Allow scaling of axes of Y.
symmetric	Use symmetric Procrustes statistic (the rotation will still be non-symmetric).
scores	Kind of scores used. This is the display argument used with the corresponding scores function: see scores, scores.cca and scores.cca for alternatives.
x, object	An object of class procrustes.
digits	Number of digits in the output.
kind	For plot function, the kind of plot produced: kind = 1 plots shifts in two configurations, kind = 0 draws a corresponding empty plot, and kind = 2 plots an impulse diagram of residuals.
choices	Axes (dimensions) plotted.
xlab, ylab	Axis labels, if defaults unacceptable.
main	Plot title, if default unacceptable.
display	Show only the "target" or "rotated" matrix as points.
to.target	Draw arrows to point to target.
type	The type of plot drawn. In plot, the type can be "points" or "text" to select the marker for the tail of the arrow, or "none" for drawing an empty plot. In lines the type selects either arrows or line segments to connect target and rotated configuration.
truemean	Use the original range of target matrix instead of centring the fitted values. Function plot.procrustes needs truemean = FALSE, and adding graphical items to the plots from the original results may need truemean = TRUE.
newdata	Matrix of coordinates to be rotated and translated to the target.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
ar.col	Arrow colour.
length	Width of the arrow head.
labels	Character vector of text labels. Rownames of the result object are used as default.
cex	Character expansion for points or text.
...	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 if to.target = TRUE, and to rotated configuration if to.target = FALSE. 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 predict can be used to add new rotated coordinates to the target. The predict function will always translate coordinates to the original non-centred matrix. The function cannot be used with newdata for symmetric analysis.

Function protest performs symmetric Procrustes analysis 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 also prints the sum of squares of the symmetric analysis, sometimes called m_{12}^2. 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.
xmean	The centroid of the target.

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. The distribution of these correlations can be inspected with permustats function.
signif	Significance of t
permutations	Number of permutations.
control	A list of control values for the permutations as returned by the function how.
control	the list passed to argument control describing the permutation design.

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

monoMDS, for obtaining objects for procrustes, and mantel for an alternative to protest without need of dimension reduction. See how for details on specifying the type of permutation required.

Examples

```
## IGNORE_RDIFF_BEGIN
data(varespec)
vare.dist <- vegdist(wisconsin(varespec))
mds.null <- monoMDS(vare.dist, y = cmdscale(vare.dist))
mds.alt <- monoMDS(vare.dist)
vare.proc <- procrustes(mds.alt, mds.null)
vare.proc
summary(vare.proc)
plot(vare.proc)
plot(vare.proc, kind=2)
residuals(vare.proc)
## IGNORE_RDIFF_END
```

pyrifos *Response of Aquatic Invertebrates to Insecticide Treatment*

Description

The data are log transformed abundances of aquatic invertebrate in twelve ditches studied in eleven times before and after an insecticide treatment.

Usage

```
data(pyrifos)
```

Format

A data frame with 132 observations on the log-transformed (`log(10*x + 1)`) abundances of 178 species. There are only twelve sites (ditches, mesocosms), but these were studied repeatedly in eleven occasions. The treatment levels, treatment times, or ditch ID's are not in the data frame, but the data are very regular, and the example below shows how to obtain these external variables.

Details

This data set was obtained from an experiment in outdoor experimental ditches. Twelve mesocosms were allocated at random to treatments; four served as controls, and the remaining eight were treated once with the insecticide chlorpyrifos, with nominal dose levels of 0.1, 0.9, 6, and 44 μg/ L in two mesocosms each. The example data set invertebrates. Sampling was done 11 times, from week -4 pre-treatment through week 24 post-treatment, giving a total of 132 samples (12 mesocosms times 11 sampling dates), see van den Brink & ter Braak (1999) for details. The data set contains only the species data, but the example below shows how to obtain the treatment, time and ditch ID variables.

Source

CANOCO 4 example data, with the permission of Cajo J. F. ter Braak.

References

van den Brink, P.J. & ter Braak, C.J.F. (1999). Principal response curves: Analysis of time-dependent multivariate responses of biological community to stress. Environmental Toxicology and Chemistry, 18, 138–148.

Examples

```
data(pyrifos)
ditch <- gl(12, 1, length=132)
week <- gl(11, 12, labels=c(-4, -1, 0.1, 1, 2, 4, 8, 12, 15, 19, 24))
dose <- factor(rep(c(0.1, 0, 0, 0.9, 0, 44, 6, 0.1, 44, 0.9, 0, 6), 11))
```

radfit *Rank – Abundance or Dominance / Diversity Models*

Description

Functions construct rank – abundance or dominance / diversity or Whittaker plots and fit broken-stick, preemption, log-Normal, Zipf and Zipf-Mandelbrot models of species abundance.

Usage

```
## Default S3 method:
radfit(x, ...)
rad.null(x, family=poisson, ...)
rad.preempt(x, family = poisson, ...)
rad.lognormal(x, family = poisson, ...)
rad.zipf(x, family = poisson, ...)
rad.zipfbrot(x, family = poisson, ...)
## S3 method for class 'radline'
predict(object, newdata, total, ...)
## S3 method for class 'radfit'
plot(x, BIC = FALSE, legend = TRUE, ...)
## S3 method for class 'radfit.frame'
plot(x, which, BIC = FALSE, model, legend = TRUE, ...)
## S3 method for class 'radline'
plot(x, xlab = "Rank", ylab = "Abundance", type = "b", ...)
radlattice(x, BIC = FALSE, ...)
## S3 method for class 'radfit'
lines(x, ...)
## S3 method for class 'radfit'
points(x, ...)
as.rad(x)
## S3 method for class 'rad'
plot(x, xlab = "Rank", ylab = "Abundance", log = "y", ...)
```

Arguments

`x`	Data frame, matrix or a vector giving species abundances, or an object to be plotted.
`family`	Error distribution (passed to `glm`). All alternatives accepting `link = "log"` in `family` can be used, although not all make sense.
`object`	A fitted result object.
`newdata`	Ranks used for ordinations. All models can interpolate to non-integer "ranks" (although this may be approximate), but extrapolation may fail
`total`	The new total used for predicting abundance. Observed total count is used if this is omitted.

`which`	A vector for selecting sites in plots.
`BIC`	Use Bayesian Information Criterion, BIC, instead of Akaike's AIC. The penalty in BIC 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 `Null`, `Preemption`, `Lognormal`, `Zipf`, `Mandelbrot`.
`legend`	Add legend of line colours.
`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.
`log`	Use logarithmic scale for given axis. The default `log = "y"` gives the traditional plot of community ecology where the preemption model is a straight line, and with `log = "xy"` Zipf model is a straight line. With `log = ""` both axes are in the original arithmetic scale.
`...`	Other parameters to functions.

Details

Rank–Abundance Dominance (RAD) or Dominance/Diversity plots (Whittaker 1965) display logarithmic species abundances against species rank order. These plots are supposed to be effective in analysing types of abundance distributions in communities. These functions fit some of the most popular models mainly following Wilson (1991).

Functions `rad.null`, `rad.preempt`, `rad.lognormal`, `rad.zipf` and `zipfbrot` fit the individual models (described below) for a single vector (row of data frame), 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.

Function `rad.null` fits a brokenstick model where the expected abundance of species at rank r is $a_r = (J/S)\sum_{x=r}^{S}(1/x)$ (Pielou 1975), where J is the total number of individuals (site total) and S is the total number of species in the community. This gives a Null model where the individuals are randomly distributed among observed species, and there are no fitted parameters. Function `rad.preempt` fits the niche preemption model, a.k.a. geometric series or Motomura model, where the expected abundance a of species at rank r is $a_r = J\alpha(1-\alpha)^{r-1}$. The only estimated parameter is the preemption coefficient α which gives the decay rate of abundance per rank. The niche preemption model is a straight line in a RAD plot. Function `rad.lognormal` fits a log-Normal model which assumes that the logarithmic abundances are distributed Normally, or $a_r = \exp(\log\mu + \log\sigma N)$, where N is a Normal deviate. Function `rad.zipf` fits the Zipf model $a_r = Jp_1r^{\gamma}$ 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.

Log-Normal and Zipf models are generalized linear models (`glm`) with logarithmic link function. Zipf–Mandelbrot adds one nonlinear parameter to the Zipf model, and is fitted using `nlm` for the nonlinear parameter and estimating other parameters and log-Likelihood with `glm`. Preemption model is fitted as a purely nonlinear model. There are no estimated parameters in the Null model.

The default `family` is `poisson` which is appropriate only for genuine counts (integers), but other families that accept `link = "log"` can be used. Families `Gamma` or `gaussian` 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.

All these functions have their own plot functions. The graph for radfit.frame will plot each sampling unit separately and you may need to divide the plot screen to panels for each of these (see par("mfrow")). The graphics functions return invisibly an ordiplot object for observed points, and function identify.ordiplot can be used to label selected species. Alternatively, radlattice uses Lattice graphics to display each radfit model of a single site in a separate panel together with their AIC or BIC values.

Function as.rad is a base function to construct ordered RAD data. Its plot is used by other RAD plot functions which pass extra arguments (such as xlab and log) to this function. The function returns an ordered vector of taxa occurring in a site, and a corresponding attribute "index" of included taxa.

Value

Functions rad.null, rad.preempt, rad.lognormal, zipf and zipfbrot fit each a single RAD model to a single site. The result object has class "radline" and inherits from glm, and can be handled by some (but not all) glm methods.

Function radfit fits all models either to a single site or to all rows of a data frame or a matrix. When fitted to a single site, the function returns an object of class "radfit" with items y (observed values), family, and models which is a list of fitted "radline" models. When applied for a data frame or matrix, radfit function returns an object of class "radfit.frame" which is a list of "radfit" objects, each item names by the corresponding row name.

All result objects ("radline", "radfit", "radfit.frame") can be accessed with same method functions. The following methods are available: AIC, coef, deviance, logLik. In addition the fit results can be accessed with fitted, predict and residuals (inheriting from residuals.glm). The graphical functions were discussed above in Details.

Note

The RAD models are usually fitted for proportions instead of original abundances. However, nothing in these models seems to require division of abundances by site totals, and original observations are used in these functions. If you wish to use proportions, you must standardize your data by site totals, e.g. with decostand and use appropriate family such as Gamma.

The lognormal model is fitted in a standard way, but I do think this is not quite correct – at least it is not equivalent to fitting Normal density to log abundances like originally suggested (Preston 1948).

Some models may fail. In particular, estimation of the Zipf-Mandelbrot model is difficult. If the fitting fails, NA is returned.

Wilson (1991) defined preemption model as $a_r = Jp_1(1-\alpha)^{r-1}$, where p_1 is the fitted proportion of the first species. However, parameter p_1 is completely defined by α since the fitted proportions must add to one, and therefore I handle preemption as a one-parameter model.

Veiled log-Normal model was included in earlier releases of this function, but it was removed because it was flawed: an implicit veil line also appears in the ordinary log-Normal. The latest release version with rad.veil was 1.6-10.

Author(s)

Jari Oksanen

References

Pielou, E.C. (1975) *Ecological Diversity*. Wiley & Sons.

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

Whittaker, R. H. (1965) Dominance and diversity in plant communities. *Science* 147, 250–260.

Wilson, J. B. (1991) Methods for fitting dominance/diversity curves. *Journal of Vegetation Science* 2, 35–46.

See Also

`fisherfit` and `prestonfit`. An alternative approach is to use `qqnorm` or `qqplot` with any distribution. For controlling graphics: `Lattice`, `xyplot`, `lset`.

Examples

```
data(BCI)
mod <- rad.lognormal(BCI[5,])
mod
plot(mod)
mod <- radfit(BCI[1,])
## Standard plot overlaid for all models
## Preemption model is a line
plot(mod)
## log-log: Zipf model is a line
plot(mod, log = "xy")
# Take a subset of BCI to save time and nerves
mod <- radfit(BCI[3:6,])
mod
op <- par(mfrow=c(2,2)) # need 4 panels
plot(mod, pch=".")
par(op)
```

---

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",
   metric = c("euclidean", "mahalanobis", "manhattan", "gower"),
   ...)
```

Arguments

grad	The gradient variable or matrix.
veg	The community data matrix.
indices	Dissimilarity indices compared, partial matches to alternatives in vegdist. Alternatively, it can be a (named) list of functions returning objects of class 'dist'.
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.
metric	Metric to evaluate the gradient separation. See Details.
...	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. The gradient separation between each point is assessed using given metric. The default is to use Euclidean distance of continuous variables scaled to unit variance, or to use Gower metric for mixed data using function daisy when grad has factors. The other alternatives are Mahalanabis distances which are based on grad matrix scaled so that columns are orthogonal (uncorrelated) and have unit variance, or Manhattan distances of grad variables scaled to unit range.

The indices argument can accept any dissimilarity indices besides the ones calculated by the vegdist function. For this, the argument value should be a (possibly named) list of functions. Each function must return a valid 'dist' object with dissimilarities, similarities are not accepted and should be converted into dissimilarities beforehand.

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.

The functions returning dissimilarity objects should be self contained, because the ... argument passes additional parameters to stepacross and not to the functions supplied via the indices argument.

Author(s)

Jari Oksanen, with additions from Peter Solymos

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, monoMDS, cor, Machine, and for alternatives anosim, mantel and protest.

Examples

```
data(varespec)
data(varechem)
## The variables are automatically scaled
rankindex(varechem, varespec)
rankindex(varechem, wisconsin(varespec))
## Using non vegdist indices as functions
funs <- list(Manhattan=function(x) dist(x, "manhattan"),
    Gower=function(x) cluster:::daisy(x, "gower"),
    Ochiai=function(x) designdist(x, "1-J/sqrt(A*B)"))
rankindex(scale(varechem), varespec, funs)
```

---

rarefy — *Rarefaction Species Richness*

Description

Rarefied species richness for community ecologists.

Usage

```
rarefy(x, sample, se = FALSE, MARGIN = 1)
rrarefy(x, sample)
drarefy(x, sample)
rarecurve(x, step = 1, sample, xlab = "Sample Size", ylab = "Species",
          label = TRUE, col, lty, tidy = FALSE, ...)
rareslope(x, sample)
```

Arguments

x	Community data, a matrix-like object or a vector.
MARGIN	Margin for which the index is computed.
sample	Subsample size for rarefying community, either a single value or a vector.

se	Estimate standard errors.
step	Step size for sample sizes in rarefaction curves.
xlab, ylab	Axis labels in plots of rarefaction curves.
label	Label rarefaction curves by rownames of x (logical).
col, lty	plotting colour and line type, see par. Can be a vector of length nrow(x), one per sample, and will be extended to such a length internally.
tidy	Instead of drawing a plot, return a "tidy" data frame than can be used in **ggplot2** graphics. The data frame has variables Site (factor), Sample and Species.
...	Parameters passed to nlm, or to plot, lines and ordilabel in rarecurve.

Details

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 work for larger sample as well (with a warning) and return non-rarefied species richness (and standard error = 0). If sample is a vector, rarefaction of all observations is performed for each sample size separately. Rarefaction can be performed only with genuine counts of individuals. The function rarefy is based on Hurlbert's (1971) formulation, and the standard errors on Heck et al. (1975).

Function rrarefy generates one randomly rarefied community data frame or vector of given sample size. The sample can be a vector giving the sample sizes for each row. If the sample size is equal to or larger than the observed number of individuals, the non-rarefied community will be returned. The random rarefaction is made without replacement so that the variance of rarefied communities is rather related to rarefaction proportion than to the size of the sample. Random rarefaction is sometimes used to remove the effects of different sample sizes. This is usually a bad idea: random rarefaction discards valid data, introduces random error and reduces the quality of the data (McMurdie & Holmes 2014). It is better to use normalizing transformations (decostand in **vegan**) possible with variance stabilization (decostand and dispweight in **vegan**) and methods that are not sensitive to sample sizes.

Function drarefy returns probabilities that species occur in a rarefied community of size sample. The sample can be a vector giving the sample sizes for each row. If the sample is equal to or larger than the observed number of individuals, all observed species will have sampling probability 1.

Function rarecurve draws a rarefaction curve for each row of the input data. The rarefaction curves are evaluated using the interval of step sample sizes, always including 1 and total sample size. If sample is specified, a vertical line is drawn at sample with horizontal lines for the rarefied species richnesses.

Function rareslope calculates the slope of rarecurve (derivative of rarefy) at given sample size; the sample need not be an integer.

Rarefaction functions should be used for observed counts. If you think it is necessary to use a multiplier to data, rarefy first and then multiply. Removing rare species before rarefaction can also give biased results. Observed count data normally include singletons (species with count 1), and if these are missing, functions issue warnings. These may be false positives, but it is recommended to check that the observed counts are not multiplied or rare taxa are not removed.

Value

A vector of rarefied species richness values. With a single `sample` and `se = TRUE`, function `rarefy` returns a 2-row matrix with rarefied richness (`S`) and its standard error (`se`). If `sample` is a vector in `rarefy`, the function returns a matrix with a column for each `sample` size, and if `se = TRUE`, rarefied richness and its standard error are on consecutive lines.

Function `rarecurve` returns `invisible` list of `rarefy` results corresponding each drawn curve. Alternatively, with `tidy = TRUE` it returns a data frame that can be used in **ggplot2** graphics.

Author(s)

Jari Oksanen

References

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

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

McMurdie, P.J. & Holmes, S. (2014). Waste not, want not: Why rarefying microbiome data is inadmissible. *PLoS Comput Biol* **10(4):** e1003531. doi:10.1371/journal.pcbi.1003531

See Also

Use `specaccum` for species accumulation curves where sites are sampled instead of individuals. `specpool` extrapolates richness to an unknown sample size.

Examples

```
data(BCI)
S <- specnumber(BCI) # observed number of species
(raremax <- min(rowSums(BCI)))
Srare <- rarefy(BCI, raremax)
plot(S, Srare, xlab = "Observed No. of Species", ylab = "Rarefied No. of Species")
abline(0, 1)
rarecurve(BCI, step = 20, sample = raremax, col = "blue", cex = 0.6)
```

---

raupcrick — *Raup-Crick Dissimilarity with Unequal Sampling Densities of Species*

Description

Function finds the Raup-Crick dissimilarity which is a probability of number of co-occurring species with species occurrence probabilities proportional to species frequencies.

Usage

```
raupcrick(comm, null = "r1", nsimul = 999, chase = FALSE, ...)
```

Arguments

comm	Community data which will be treated as presence/absence data.
null	Null model used as the method in oecosimu.
nsimul	Number of null communities for assessing the dissimilarity index.
chase	Use the Chase et al. (2011) method of tie handling (not recommended except for comparing the results against the Chase script).
...	Other parameters passed to oecosimu.

Details

Raup-Crick index is the probability that compared sampling units have non-identical species composition. This probability can be regarded as a dissimilarity, although it is not metric: identical sampling units can have dissimilarity slightly above 0, the dissimilarity can be nearly zero over a range of shared species, and sampling units with no shared species can have dissimilarity slightly below 1. Moreover, communities sharing rare species appear as more similar (lower probability of finding rare species together), than communities sharing the same number of common species.

The function will always treat the data as binary (presence/ absence).

The probability is assessed using simulation with oecosimu where the test statistic is the observed number of shared species between sampling units evaluated against a community null model (see Examples). The default null model is "r1" where the probability of selecting species is proportional to the species frequencies.

The vegdist function implements a variant of the Raup-Crick index with equal sampling probabilities for species using exact analytic equations without simulation. This corresponds to null model "r0" which also can be used with the current function. All other null model methods of oecosimu can be used with the current function, but they are new unpublished methods.

Value

The function returns an object inheriting from dist which can be interpreted as a dissimilarity matrix.

Note

The test statistic is the number of shared species, and this is typically tied with a large number of simulation results. The tied values are handled differently in the current function and in the function published with Chase et al. (2011). In **vegan**, the index is the number of simulated values that are smaller *or equal* than the observed value, but smaller than observed value is used by Chase et al. (2011) with option split = FALSE in their script; this can be achieved with chase = TRUE in **vegan**. Chase et al. (2011) script with split = TRUE uses half of tied simulation values to calculate a distance measure, and that choice cannot be directly reproduced in vegan (it is the average of **vegan** raupcrick results with chase = TRUE and chase = FALSE).

Author(s)

The function was developed after Brian Inouye contacted us and informed us about the method in Chase et al. (2011), and the function takes its idea from the code that was published with their paper. The current function was written by Jari Oksanen.

References

Chase, J.M., Kraft, N.J.B., Smith, K.G., Vellend, M. and Inouye, B.D. (2011). Using null models to disentangle variation in community dissimilarity from variation in α-diversity. *Ecosphere* 2:art24 doi:10.1890/ES1000117.1

See Also

The function is based on `oecosimu`. Function `vegdist` with `method = "raup"` implements a related index but with equal sampling densities of species, and `designdist` demonstrates its calculation.

Examples

```
## data set with variable species richness
data(sipoo)
## default raupcrick
dr1 <- raupcrick(sipoo)
## use null model "r0" of oecosimu
dr0 <- raupcrick(sipoo, null = "r0")
## vegdist(..., method = "raup") corresponds to 'null = "r0"'
d <- vegdist(sipoo, "raup")
op <- par(mfrow=c(2,1), mar=c(4,4,1,1)+.1)
plot(dr1 ~ d, xlab = "Raup-Crick with Null R1", ylab="vegdist")
plot(dr0 ~ d, xlab = "Raup-Crick with Null R0", ylab="vegdist")
par(op)

## The calculation is essentially as in the following oecosimu() call,
## except that designdist() is replaced with faster code
## Not run:
oecosimu(sipoo, function(x) designdist(x, "J", "binary"), method = "r1")

## End(Not run)
```

`read.cep` *Reads a CEP (Canoco) data file*

Description

`read.cep` reads a file formatted with relaxed strict CEP format used in Canoco software, among others.

Usage

```
read.cep(file, positive=TRUE)
```

Arguments

`file`	File name (character variable).
`positive`	Only positive entries, like in community data.

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. A relaxed variant of this format was later adopted in Canoco software (ter Braak 1984). Function `read.cep` reads legacy files written in this format.

The condensed CEP and CANOCO formats have:

- Two or three title cards, most importantly specifying the format and the number of items per record.
- Data in condensed format: First number on the line is the site identifier (an integer), and it is followed by pairs ('couplets') of numbers identifying the species and its abundance (an integer and a floating point number).
- 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 rows 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

Function `read.cep` used Fortran to read data in **vegan** 2.4-5 and earlier, but Fortran I/O is no longer allowed in CRAN packages, and the function was re-written in R. The original Fortran code was more robust, and there are several legacy data sets that may fail with the current version, but could be read with the previous Fortran version. CRAN package **cepreader** makes available the original Fortran-based code run in a separate subprocess. The **cepreader** package can also read 'free' and 'open' Canoco formats that are not handled in this function.

The function is based on `read.fortran`. If the `REAL` format defines a decimal part for species abundances (such as `F5.1`), `read.fortran` divides the input with the corresponding power of 10 even when the input data had explicit decimal separator. With `F5.1`, 100 would become 10, and 0.1 become 0.01. Function `read.cep` tries to undo this division, but you should check the scaling of results after reading the data, and if necessary, multiply results to the original scale.

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")
## End(Not run)
```

---

renyi *Renyi and Hill Diversities and Corresponding Accumulation Curves*

Description

Function `renyi` find Rényi diversities with any scale or the corresponding Hill number (Hill 1973). Function `renyiaccum` finds these statistics with accumulating sites.

Usage

```
renyi(x, scales = c(0, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, Inf),
   hill = FALSE)
renyiaccum(x, scales = c(0, 0.5, 1, 2, 4, Inf), permutations = 100,
    raw = FALSE, collector = FALSE, subset, ...)
## S3 method for class 'renyi'
plot(x, ...)
## S3 method for class 'renyiaccum'
plot(x, ...)
## S3 method for class 'renyiaccum'
lines(x, what, ...)
## S3 method for class 'renyiaccum'
persp(x, theta = 220, col = heat.colors(100), zlim, ...)
```

Arguments

`x`	Community data matrix or plotting object.
`scales`	Scales of Rényi diversity.
`hill`	Calculate Hill numbers.
`permutations`	Usually an integer giving the number permutations, but can also be a list of control values for the permutations as returned by the function `how`, or a permutation matrix where each row gives the permuted indices.
`raw`	if `FALSE` then return summary statistics of permutations, and if `TRUE` then returns the individual permutations.
`collector`	Accumulate the diversities in the order the sites are in the data set, and the collector curve can be plotted against summary of permutations. The argument is ignored if `raw = TRUE`.
`subset`	logical expression indicating sites (rows) to keep: missing values are taken as `FALSE`.
`theta`	Angle defining the viewing direction (azimuthal) in `persp`.

what	Lines to be drawn. These can be any item of dimension 3 of renyiaccum result, with defaults "mean", "min", "max", "Qnt 0.025", "Qnt 0.975", and optional "Collector" or quoted numbers of permutations.
col	Colours used for surface. Single colour will be passed on, and vector colours will be selected by the midpoint of a rectangle in persp.
zlim	Limits of vertical axis.
...	Other arguments which are passed to renyi and to graphical functions.

Details

Common diversity indices are special cases of Rényi diversity

$$H_a = \frac{1}{1-a} \log \sum p_i^a$$

where a is a scale parameter, and Hill (1975) suggested to use so-called 'Hill numbers' defined as $N_a = \exp(H_a)$. Some Hill numbers are the number of species with $a = 0$, $\exp(H')$ or the exponent of Shannon diversity with $a = 1$, inverse Simpson with $a = 2$ and $1/\max(p_i)$ with $a = \infty$. According to the theory of diversity ordering, one community can be regarded as more diverse than another only if its Rényi diversities are all higher (Tóthmérész 1995).

Function renyiaccum is similar to specaccum but finds Rényi or Hill diversities at given scales for random permutations of accumulated sites. It has a persp method to plot the diversity surface against scale and number and sites.

plot and lines methods for renyi and renyiaccum are based on matplot and matlines and draw everything on a single plot frame. In **vegan** pre version 2.8-0 these graphs were based on **lattice** graphics that used separate panels. This types of plots can be made in the **ggvegan** package (fortify functions). **vegan3d** can make dynamics graphics with rgl.renyiaccum.

Value

Function renyi returns a data frame of selected indices. Function renyiaccum with argument raw = FALSE returns a three-dimensional array, where the first dimension are the accumulated sites, second dimension are the diversity scales, and third dimension are the summary statistics mean, stdev, min, max, Qnt 0.025 and Qnt 0.975. With argument raw = TRUE the statistics on the third dimension are replaced with individual permutation results.

Author(s)

Roeland Kindt and Jari Oksanen

References

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

Kindt, R., Van Damme, P., Simons, A.J. (2006). Tree diversity in western Kenya: using profiles to characterise richness and evenness. *Biodiversity and Conservation* 15, 1253–1270.

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

See Also

diversity for diversity indices, and specaccum for ordinary species accumulation curves. For alternative graphics, see fortify function in **ggvegan** package.

Examples

```
data(BCI)
i <- sample(nrow(BCI), 12)
mod <- renyi(BCI[i,])
plot(mod)
mod <- renyiaccum(BCI)
persp(mod)
## plot with empirical 95% confidence limits
plot(mod, lwd = 3)
lines(mod, "Qnt 0.025", lty=1)
lines(mod, "Qnt 0.975", lty=1)
```

---

reorder.hclust *Reorder a Hierarchical Clustering Tree*

Description

Function takes a hierarchical clustering tree from hclust and a vector of values and reorders the clustering tree in the order of the supplied vector, maintaining the constraints on the tree. This is a method of generic function reorder and an alternative to reordering a "dendrogram" object with reorder.dendrogram

Usage

```
## S3 method for class 'hclust'
reorder(x, wts,
   agglo.FUN = c("mean", "min", "max", "sum", "uwmean"), ...)
## S3 method for class 'hclust'
rev(x)
## S3 method for class 'hclust'
scores(x, display = "internal", ...)
cutreeord(tree, k = NULL, h = NULL)
```

Arguments

x, tree	hierarchical clustering from hclust.
wts	numeric vector for reordering.
agglo.FUN	a function for weights agglomeration, see below.
display	return "internal" nodes or "terminal" nodes (also called "leaves").
k, h	scalars or vectors giving the numbers of desired groups or the heights where the tree should be cut (passed to function cutree).
...	additional arguments (ignored).

Details

Dendrograms can be ordered in many ways. The `reorder` function reorders an `hclust` tree and provides an alternative to `reorder.dendrogram` which can reorder a `dendrogram`. The current function will also work differently when the `agglo.FUN` is `"mean"`: the `reorder.dendrogram` will always take the direct mean of member groups ignoring their sizes, but this function will used `weighted.mean` weighted by group sizes, so that the group mean is always the mean of member leaves (terminal nodes). If you want to ignore group sizes, you can use unweighted mean with `"uwmean"`.

The function accepts only a limited list of `agglo.FUN` functions for assessing the value of `wts` for groups. The ordering is always ascending, but the order of leaves can be reversed with `rev`.

Function `scores` finds the coordinates of nodes as a two-column matrix. For terminal nodes (leaves) this the value at which the item is merged to the tree, and the labels can still hang below this level (see `plot.hclust`).

Function `cutreeord` cuts a tree to groups numbered from left to right in the tree. It is based on the standard function `cutree` which numbers the groups in the order they appear in the input data instead of the order in the tree.

Value

Reordered `hclust` result object with added item `value` that gives the value of the statistic at each merge level.

Note

These functions should really be in base R.

Author(s)

Jari Oksanen

See Also

`hclust` for getting clustering trees, `as.hclust.spantree` to change a **vegan** minimum spanning tree to an `hclust` object, and `dendrogram` and `reorder.dendrogram` for an alternative implementation.

Examples

```
## reorder by water content of soil
data(mite, mite.env)
hc <- hclust(vegdist(wisconsin(sqrt(mite))))
ohc <- with(mite.env, reorder(hc, WatrCont))
plot(hc)
plot(ohc)

## label leaves by the observed value, and each branching point
## (internal node) by the cluster mean
with(mite.env, plot(ohc, labels=round(WatrCont), cex=0.7))
ordilabel(scores(ohc), label=round(ohc$value), cex=0.7)
```

```
## Slightly different from reordered 'dendrogram' which ignores group
## sizes in assessing means.
den <- as.dendrogram(hc)
den <- with(mite.env, reorder(den, WatrCont, agglo.FUN = mean))
plot(den)
```

RsquareAdj *Adjusted R-square*

Description

The functions finds the adjusted R-square.

Usage

```
## Default S3 method:
RsquareAdj(x, n, m, ...)
## S3 method for class 'rda'
RsquareAdj(x, ...)
## S3 method for class 'cca'
RsquareAdj(x, permutations = 1000, ...)
```

Arguments

x	Unadjusted R-squared or an object from which the terms for evaluation or adjusted R-squared can be found.
n, m	Number of observations and number of degrees of freedom in the fitted model.
permutations	Number of permutations to use when computing the adjusted R-squared for a cca. The permutations can be calculated in parallel by specifying the number of cores which is passed to permutest
...	Other arguments (ignored) except in the case of cca in which these arguments are passed to permutest.

Details

The default method finds the adjusted R^2 from the unadjusted R^2, number of observations, and number of degrees of freedom in the fitted model. The specific methods find this information from the fitted result object. There are specific methods for rda (also used for distance-based RDA), cca, lm and glm. Adjusted, or even unadjusted, R^2 may not be available in some cases, and then the functions will return NA. R^2 values are available only for gaussian models in glm.

The adjusted, R^2 of cca is computed using a permutation approach developed by Peres-Neto et al. (2006). By default 1000 permutations are used.

Value

The functions return a list of items r.squared and adj.r.squared.

References

Legendre, P., Oksanen, J. and ter Braak, C.J.F. (2011). Testing the significance of canonical axes in redundancy analysis. *Methods in Ecology and Evolution* 2, 269–277.

Peres-Neto, P., P. Legendre, S. Dray and D. Borcard. 2006. Variation partitioning of species data matrices: estimation and comparison of fractions. *Ecology* 87, 2614–2625.

See Also

varpart uses RsquareAdj.

Examples

```
data(mite)
data(mite.env)
## rda
m <- rda(decostand(mite, "hell") ~  ., mite.env)
RsquareAdj(m)
## cca
m <- cca(decostand(mite, "hell") ~  ., mite.env)
RsquareAdj(m)
## default method
RsquareAdj(0.8, 20, 5)
```

---

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. The scores function is generic in **vegan**, and **vegan** ordination functions have their own scores functions that are documented separately with the method (see e.g. scores.cca, scores.metaMDS, scores.decorana). This help file documents the default scores method that is only used for non-**vegan** ordination objects.

Usage

```
## Default S3 method:
scores(x, choices,
    display=c("sites", "species", "both"), tidy = FALSE, ...)
```

Arguments

x	An ordination result.
choices	Ordination axes. If missing, default method returns all axes.
display	Partial match to access scores for "sites" or "species" of for "both".
tidy	Return "both" scores in data frame that is compatible with **ggvegan**, with variable score labelling the scores as "sites" or "species".
...	Other parameters (unused).

Details

Function scores is a generic method in **vegan**. Several **vegan** functions have their own scores methods with their own defaults and with some new arguments. This help page describes only the default method. For other methods, see, e.g., scores.cca, scores.rda, scores.decorana.

All **vegan** ordination functions should have a scores method which should be used to extract the scores instead of directly accessing them. Scaling and transformation of scores should also happen in the scores function. If the scores function is available, the results can be plotted using ordiplot, ordixyplot etc., and the ordination results can be compared in procrustes analysis.

The scores.default function is used to extract scores from non-**vegan** ordination results. Many standard ordination methods of libraries 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.

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 and some **ade4** objects. It may work in other cases or fail mysteriously.

Value

The function returns a matrix of scores if one type is requested, or a named list of matrices if display = "both", or a **ggvegan** compatible data frame if tidy = TRUE.

Author(s)

Jari Oksanen

See Also

Specific scores functions include (but are not limited to) scores.cca, scores.rda, scores.decorana, scores.envfit, scores.metaMDS, scores.monoMDS and scores.pcnm. These have somewhat different interface – scores.cca in particular – but all work with keywords display="sites" and return a matrix. However, they may also return a list of matrices, and some other scores methods will have quite different arguments.

Examples

```
data(varespec)
vare.pca <- prcomp(varespec)
scores(vare.pca, choices=c(1,2))
```

---

screeplot.cca *Screeplots for Ordination Results and Broken Stick Distributions*

Description

Screeplot methods for plotting variances of ordination axes/components and overlaying broken stick distributions. Also, provides alternative screeplot methods for princomp and prcomp.

Usage

```
## S3 method for class 'cca'
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
      npcs = min(10, if (is.null(x$CCA) || x$CCA$rank == 0) x$CA$rank else x$CCA$rank),
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse1(substitute(x)), legend = bstick,
         ...)

## S3 method for class 'decorana'
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = 4,
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse1(substitute(x)), legend = bstick,
         ...)

## S3 method for class 'prcomp'
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = min(10, length(x$sdev)),
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse1(substitute(x)), legend = bstick,
         ...)

## S3 method for class 'princomp'
screeplot(x, bstick = FALSE, type = c("barplot", "lines"),
         npcs = min(10, length(x$sdev)),
         ptype = "o", bst.col = "red", bst.lty = "solid",
         xlab = "Component", ylab = "Inertia",
         main = deparse1(substitute(x)), legend = bstick,
         ...)

bstick(n, ...)

## Default S3 method:
bstick(n, tot.var = 1, ...)

## S3 method for class 'cca'
bstick(n, ...)

## S3 method for class 'prcomp'
bstick(n, ...)

## S3 method for class 'princomp'
bstick(n, ...)

## S3 method for class 'decorana'
```

```
bstick(n, ...)
```

Arguments

x	an object from which the component variances can be determined.
bstick	logical; should the broken stick distribution be drawn?
npcs	the number of components to be plotted.
type	the type of plot.
ptype	if type == "lines" or bstick = TRUE, a character indicating the type of plotting used for the lines; actually any of the types as in plot.default.
bst.col, bst.lty	the colour and line type used to draw the broken stick distribution.
xlab, ylab, main	graphics parameters.
legend	logical; draw a legend?
n	an object from which the variances can be extracted or the number of variances (components) in the case of bstick.default.
tot.var	the total variance to be split.
...	arguments passed to other methods.

Details

The functions provide screeplots for most ordination methods in **vegan** and enhanced versions with broken stick for prcomp and princomp.

Function bstick gives the brokenstick values which are ordered random proportions, defined as $p_i = (tot/n) \sum_{x=i}^{n} (1/x)$ (Legendre & Legendre 2012), where tot is the total and n is the number of brokenstick components (cf. radfit). Broken stick has been recommended as a stopping rule in principal component analysis (Jackson 1993): principal components should be retained as long as observed eigenvalues are higher than corresponding random broken stick components.

The bstick function is generic. The default needs the number of components and the total, and specific methods extract this information from ordination results. There also is a bstick method for cca. However, the broken stick model is not strictly valid for correspondence analysis (CA), because eigenvalues of CA are defined to be ≤ 1, whereas brokenstick components have no such restrictions. The brokenstick components in detrended correspondence analysis (DCA) assume that input data are of full rank, and additive eigenvalues are used in screeplot (see decorana).

Value

Function screeplot draws a plot on the currently active device, and returns invisibly the xy.coords of the points or bars for the eigenvalues.

Function bstick returns a numeric vector of broken stick components.

Author(s)

Gavin L. Simpson

References

Jackson, D. A. (1993). Stopping rules in principal components analysis: a comparison of heuristical and statistical approaches. *Ecology* 74, 2204–2214.

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English ed. Elsevier.

See Also

cca, decorana, princomp and prcomp for the ordination functions, and screeplot for the stock version.

Examples

```
data(varespec)
vare.pca <- rda(varespec, scale = TRUE)
bstick(vare.pca)
screeplot(vare.pca, bstick = TRUE, type = "lines")
```

---

simper *Similarity Percentages*

Description

Discriminating species between two groups using Bray-Curtis dissimilarities

Usage

```
simper(comm, group, permutations = 999, parallel = 1, ...)
## S3 method for class 'simper'
summary(object, ordered = TRUE,
    digits = max(3,getOption("digits") - 3), ...)
```

Arguments

comm	Community data.
group	Factor describing the group structure. If this is missing or has only one level, contributions are estimated for non-grouped data and dissimilarities only show the overall heterogeneity in species abundances.
permutations	a list of control values for the permutations as returned by the function how, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
object	an object returned by simper.
ordered	Logical; Should the species be ordered by their average contribution?
digits	Number of digits in output.
parallel	Number of parallel processes or a predefined socket cluster. With parallel = 1 uses ordinary, non-parallel processing. (Not yet implemented).
...	Parameters passed to other functions. In simper the extra parameters are passed to shuffleSet if permutations are used.

Details

Similarity percentage, `simper` (Clarke 1993) is based on the decomposition of Bray-Curtis dissimilarity index (see `vegdist`, `designdist`). The contribution of individual species i to the overall Bray-Curtis dissimilarity d_{jk} is given by

$$d_{ijk} = \frac{|x_{ij} - x_{ik}|}{\sum_{i=1}^{S}(x_{ij} + x_{ik})}$$

where x is the abundance of species i in sampling units j and k. The overall index is the sum of the individual contributions over all S species $d_{jk} = \sum_{i=1}^{S} d_{ijk}$.

The `simper` functions performs pairwise comparisons of groups of sampling units and finds the contribution of each species to the average between-group Bray-Curtis dissimilarity. Although the method is called "Similarity Percentages", it really studied dissimilarities instead of similarities (Clarke 1993).

The function displays most important species for each pair of `groups`. These species contribute at least to 70 % of the differences between groups. The function returns much more extensive results (including all species) which can be accessed directly from the result object (see section Value). Function `summary` transforms the result to a list of data frames. With argument `ordered = TRUE` the data frames also include the cumulative contributions and are ordered by species contribution.

The results of `simper` can be very difficult to interpret and they are often misunderstood even in publications. The method gives the contribution of each species to overall dissimilarities, but these are caused by variation in species abundances, and only partly by differences among groups. Even if you make groups that are copies of each other, the method will single out species with high contribution, but these are not contributions to non-existing between-group differences but to random noise variation in species abundances. The most abundant species usually have highest variances, and they have high contributions even when they do not differ among groups. Permutation tests study the differences among groups, and they can be used to find out the species for which the differences among groups is an important component of their contribution to dissimilarities. Analysis without `group` argument will find species contributions to the average overall dissimilarity among sampling units. These non-grouped contributions can be compared to grouped contributions to see how much added value the grouping has for each species.

Value

A list of class `"simper"` with following items:

`species`	The species names.
`average`	Species contribution to average between-group dissimilarity.
`overall`	The average between-group dissimilarity. This is the sum of the item `average`.
`sd`	Standard deviation of contribution.
`ratio`	Average to sd ratio.
`ava`, `avb`	Average abundances per group.
`ord`	An index vector to order vectors by their contribution or order `cusum` back to the original data order.
`cusum`	Ordered cumulative contribution. These are based on item `average`, but they sum up to total 1.

p — Permutation p-value. Probability of getting a larger or equal average contribution in random permutation of the group factor. These area only available if `permutations` were used (default: not calculated).

Author(s)

Eduard Szöcs and Jari Oksanen.

References

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

See Also

Function `meandist` shows the average between-group dissimilarities (as well as the within-group dissimilarities).

Examples

```
data(dune)
data(dune.env)
(sim <- with(dune.env, simper(dune, Management, permutations = 99)))
## IGNORE_RDIFF_BEGIN
summary(sim)
## IGNORE_RDIFF_END
```

`simulate.rda` *Simulate Responses with Gaussian Error or Permuted Residuals for Constrained Ordination*

Description

Function simulates a response data frame so that it adds Gaussian error to the fitted responses of Redundancy Analysis (`rda`), Constrained Correspondence Analysis (`cca`) or distance-based RDA (`capscale`). The function is a special case of generic `simulate`, and works similarly as `simulate.lm`.

Usage

```
## S3 method for class 'rda'
simulate(object, nsim = 1, seed = NULL, indx = NULL,
    rank = "full", correlated = FALSE, ...)
```

Arguments

`object`	an object representing a fitted `rda`, `cca` or `capscale` model.
`nsim`	number of response matrices to be simulated. Only one dissimilarity matrix is returned for `capscale`, and larger `nsim` is an error.
`seed`	an object specifying if and how the random number generator should be initialized ('seeded'). See `simulate` for details.
`indx`	Index of residuals added to the fitted values, such as produced by `shuffleSet` or `sample`. The index can have duplicate entries so that bootstrapping is allowed. If `nsim` > 1, the output should be compliant with `shuffleSet` with one line for each simulation. If `nsim` is missing, the number of rows of `indx` is used to define the number of simulations, but if `nsim` is given, it should match number of rows in `indx`. If null, parametric simulation is used and Gaussian error is added to the fitted values.
`rank`	The rank of the constrained component: passed to `predict.rda` or `predict.cca`.
`correlated`	Are species regarded as correlated in parametric simulation or when `indx` is not given? If `correlated = TRUE`, multivariate Gaussian random error is generated, and if `FALSE`, Gaussian random error is generated separately for each species. The argument has no effect in `capscale` which has no information on species.
`...`	additional optional arguments (ignored).

Details

The implementation follows `"lm"` method of `simulate`, and adds Gaussian (Normal) error to the fitted values (`fitted.rda`) using function `rnorm` if `correlated = FALSE` or `mvrnorm` if `correlated = TRUE`. The standard deviations (`rnorm`) or covariance matrices for species (`mvrnorm`) are estimated from the residuals after fitting the constraints. Alternatively, the function can take a permutation index that is used to add permuted residuals (unconstrained component) to the fitted values. Raw data are used in `rda`. Internal Chi-square transformed data are used in `cca` within the function, but the returned matrix is similar to the original input data. The simulation is performed on internal metric scaling data in `capscale`, but the function returns the Euclidean distances calculated from the simulated data. The simulation uses only the real components, and the imaginary dimensions are ignored.

Value

If `nsim = 1`, returns a matrix or dissimilarities (in `capscale`) with similar additional arguments on random number seed as `simulate`. If `nsim > 1`, returns a similar array as returned by `simulate.nullmodel` with similar attributes.

Author(s)

Jari Oksanen

See Also

`simulate` for the generic case and for `lm` objects, and `simulate.nullmodel` for community null model simulation. Functions `fitted.rda` and `fitted.cca` return fitted values without the error component. See `rnorm` and `mvrnorm` (**MASS** package) for simulating Gaussian random error.

Examples

```
data(dune)
data(dune.env)
mod <- rda(dune ~  Moisture + Management, dune.env)
## One simulation
update(mod, simulate(mod) ~  .)
## An impression of confidence regions of site scores
plot(mod, display="sites")
for (i in 1:5) lines(procrustes(mod, update(mod, simulate(mod) ~ .)), col="blue")
## Simulate a set of null communities with permutation of residuals
simulate(mod, indx = shuffleSet(nrow(dune), 99))
```

`sipoo` *Birds in the Archipelago of Sipoo (Sibbo and Borgå)*

Description

Land birds on islands covered by coniferous forest in the Sipoo Archipelago, southern Finland.

Usage

```
data(sipoo)
data(sipoo.map)
```

Format

The `sipoo` data frame contains data of occurrences of 50 land bird species on 18 islands in the Sipoo Archipelago (Simberloff & Martin, 1991, Appendix 3). The species are referred by 4+4 letter abbreviation of their Latin names (but using five letters in two species names to make these unique).

The `sipoo.map` data contains the geographic coordinates of the islands in the ETRS89-TM35FIN coordinate system (EPSG:3067) and the areas of islands in hectares.

Source

Simberloff, D. & Martin, J.-L. (1991). Nestedness of insular avifaunas: simple summary statistics masking complex species patterns. *Ornis Fennica* 68:178–192.

Examples

```
data(sipoo)
data(sipoo.map)
plot(N ~ E, data=sipoo.map, asp = 1)
```

spantree *Minimum Spanning Tree*

Description

Function spantree finds a minimum spanning tree connecting all points, but disregarding dissimilarities that are at or above the threshold or NA.

Usage

```
spantree(d, toolong = 0)
## S3 method for class 'spantree'
as.hclust(x, ...)
## S3 method for class 'spantree'
cophenetic(x)
spandepth(x)
## S3 method for class 'spantree'
plot(x, ord, cex = 0.7, type = "p", labels, dlim,
     FUN = sammon,  ...)
## S3 method for class 'spantree'
lines(x, ord, display="sites", col = 1, ...)
```

Arguments

d	Dissimilarity data inheriting from class dist or a an object, such as a matrix, that can be converted to a dissimilarity matrix. Functions vegdist and dist are some functions producing suitable dissimilarity data.
toolong	Shortest dissimilarity regarded as NA. The function uses a fuzz factor, so that dissimilarities close to the limit will be made NA, too. If toolong = 0 (or negative), no dissimilarity is regarded as too long.
x	A spantree result object.
ord	An ordination configuration, or an ordination result known by scores.
cex	Character expansion factor.
type	Observations are plotted as points with type="p" or type="b", or as text label with type="t". The tree (lines) will always be plotted.
labels	Text used with type="t" or node names if this is missing.
dlim	A ceiling value used to highest cophenetic dissimilarity.
FUN	Ordination function to find the configuration from cophenetic dissimilarities. If the supplied FUN does not work, supply ordination result as argument ord.
display	Type of scores used for ord.
col	Colour of line segments. This can be a vector which is recycled for points, and the line colour will be a mixture of two joined points.
...	Other parameters passed to functions.

Details

Function spantree finds a minimum spanning tree for dissimilarities (there may be several minimum spanning trees, but the function finds only one). Dissimilarities at or above the threshold toolong and 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. Connected subtrees can be identified using distconnected.

Minimum spanning tree is closely related to single linkage clustering, a.k.a. nearest neighbour clustering, and in genetics as neighbour joining tree available in hclust and agnes functions. The most important practical difference is that minimum spanning tree has no concept of cluster membership, but always joins individual points to each other. Function as.hclust can change the spantree result into a corresponding hclust object.

Function cophenetic finds distances between all points along the tree segments. Function spandepth returns the depth of each node. The nodes of a tree are either leaves (with one link) or internal nodes (more than one link). The leaves are recursively removed from the tree, and the depth is the layer at with the leaf was removed. In disconnected spantree object (in a forest) each tree is analysed separately and disconnected nodes not in any tree have depth zero.

Function plot displays the tree over a supplied ordination configuration, and lines adds a spanning tree to an ordination graph. If configuration is not supplied for plot, the function ordinates the cophenetic dissimilarities of the spanning tree and overlays the tree on this result. The default ordination function is sammon (package **MASS**), because Sammon scaling emphasizes structure in the neighbourhood of nodes and may be able to beautifully represent the tree (you may need to set dlim, and sometimes the results will remain twisted). These ordination methods do not work with disconnected trees, but you must supply the ordination configuration. Function lines will overlay the tree in an existing plot.

Function spantree uses Prim's method implemented as priority-first search for dense graphs (Sedgewick 1990). Function cophenetic uses function stepacross with option path = "extended". The spantree is very fast, but cophenetic is slow in very large data sets.

Value

Function spantree returns an object of class spantree which is a list with two vectors, each of length $n-1$. The number of links in a tree is one less the number of observations, and the first item is omitted. The items are

kid	The child node of the parent, starting from parent number two. If there is no link from the parent, value will be NA and tree is disconnected at the node.
dist	Corresponding distance. If kid = NA, then dist = 0.
labels	Names of nodes as found from the input dissimilarities.
call	The function call.

Note

In principle, minimum spanning tree is equivalent to single linkage clustering that can be performed using hclust or agnes. However, these functions combine clusters to each other and the information of the actually connected points (the "single link") cannot be recovered from the result. The graphical output of a single linkage clustering plotted with ordicluster will look very different from an equivalent spanning tree plotted with lines.spantree.

Author(s)

Jari Oksanen

References

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

See Also

vegdist or dist for getting dissimilarities, and hclust or agnes for single linkage clustering.

Examples

```
data(dune)
dis <- vegdist(dune)
tr <- spantree(dis)
## Add tree to a metric scaling
plot(tr, cmdscale(dis), type = "t")
## Find a configuration to display the tree neatly
plot(tr, type = "t")
## Depths of nodes
depths <- spandepth(tr)
plot(tr, type = "t", label = depths)
## Plot as a dendrogram
cl <- as.hclust(tr)
plot(cl)
## cut hclust tree to classes and show in colours in spantree
plot(tr, col = cutree(cl, 5), pch=16)
```

---

specaccum *Species Accumulation Curves*

Description

Function specaccum finds species accumulation curves or the number of species for a certain number of sampled sites or individuals.

Usage

```
specaccum(comm, method = "exact", permutations = 100,
          conditioned =TRUE, gamma = "jack1",  w = NULL, subset, ...)
## S3 method for class 'specaccum'
plot(x, add = FALSE, random = FALSE, ci = 2,
    ci.type = c("bar", "line", "polygon"), col = par("fg"), lty = 1,
    ci.col = col, ci.lty = 1, ci.length = 0, xlab, ylab = x$method, ylim,
    xvar = c("sites", "individuals", "effort"), ...)
## S3 method for class 'specaccum'
boxplot(x, add = FALSE, ...)
```

```
fitspecaccum(object, model, method = "random", ...)
## S3 method for class 'fitspecaccum'
plot(x, col = par("fg"), lty = 1, xlab = "Sites",
    ylab = x$method, ...)
## S3 method for class 'specaccum'
predict(object, newdata, interpolation = c("linear", "spline"), ...)
## S3 method for class 'fitspecaccum'
predict(object, newdata, ...)
specslope(object, at)
```

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". Usually an integer giving the number permutations, but can also be a list of control values for the permutations as returned by the function how, or a permutation matrix where each row gives the permuted indices.
conditioned	Estimation of standard deviation is conditional on the empirical dataset for the exact SAC
gamma	Method for estimating the total extrapolated number of species in the survey area by function specpool
w	Weights giving the sampling effort.
subset	logical expression indicating sites (rows) to keep: missing values are taken as FALSE.
x	A specaccum result object
add	Add to an existing graph.
random	Draw each random simulation separately instead of drawing their average and confidence intervals.
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.
lty	line type (see par).
ci.col	Colour for drawing lines or filling the "polygon".
ci.lty	Line type for confidence intervals or border of the "polygon".
ci.length	Length of horizontal bars (in inches) at the end of vertical bars with ci.type = "bar".
xlab, ylab	Labels for x (defaults xvar) and y axis.

ylim	the y limits of the plot.
xvar	Variable used for the horizontal axis: "individuals" can be used only with method = "rarefaction".
object	Either a community data set or fitted specaccum model.
model	Nonlinear regression model (nls). See Details.
newdata	Optional data used in prediction interpreted as number of sampling units (sites). If missing, fitted values are returned.
interpolation	Interpolation method used with newdata.
at	Number of plots where the slope is evaluated. Can be a real number.
...	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 sample-based rarefaction method that has been independently developed numerous times (Chiarucci et al. 2008) and it is often known as Mao Tau estimate (Colwell et al. 2012). The unconditional standard deviation for the exact SAC represents a moment-based estimation that is not conditioned on the empirical data set (sd for all samples > 0). The unconditional standard deviation is based on an estimation of the extrapolated number of species in the survey area (a.k.a. gamma diversity), as estimated by function specpool. The conditional standard deviation that was developed by Jari Oksanen (not published, sd=0 for all samples). 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.

Methods "random" and "collector" can take weights (w) that give the sampling effort for each site. The weights w do not influence the order the sites are accumulated, but only the value of the sampling effort so that not all sites are equal. The summary results are expressed against sites even when the accumulation uses weights (methods "random", "collector"), or is based on individuals ("rarefaction"). The actual sampling effort is given as item Effort or Individuals in the printed result. For weighted "random" method the effort refers to the average effort per site, or sum of weights per number of sites. With weighted method = "random", the averaged species richness is found from linear interpolation of single random permutations. Therefore at least the first value (and often several first) have NA richness, because these values cannot be interpolated in all cases but should be extrapolated. The plot function defaults to display the results as scaled to sites, but this can be changed selecting xvar = "effort" (weighted methods) or xvar = "individuals" (with method = "rarefaction").

The summary and boxplot methods are available for method = "random".

Function predict for specaccum can return the values corresponding to newdata. With method "exact", "rarefaction" and "coleman" the function uses analytic equations for interpolated non-integer values, and for other methods linear (approx) or spline (spline) interpolation. If newdata is not given, the function returns the values corresponding to the data. NB., the fitted values with

method="rarefaction" are based on rounded integer counts, but predict can use fractional non-integer counts with newdata and give slightly different results.

Function fitspecaccum fits a nonlinear (nls) self-starting species accumulation model. The input object can be a result of specaccum or a community in data frame. In the latter case the function first fits a specaccum model and then proceeds with fitting the nonlinear model. The function can apply a limited set of nonlinear regression models suggested for species-area relationship (Dengler 2009). All these are selfStart models. The permissible alternatives are "arrhenius" (SSarrhenius), "gleason" (SSgleason), "gitay" (SSgitay), "lomolino" (SSlomolino) of **vegan** package. In addition the following standard R models are available: "asymp" (SSasymp), "gompertz" (SSgompertz), "michaelis-menten" (SSmicmen), "logis" (SSlogis), "weibull" (SSweibull). See these functions for model specification and details.

When weights w were used the fit is based on accumulated effort and in model = "rarefaction" on accumulated number of individuals. The plot is still based on sites, unless other alternative is selected with xvar.

Function predict for fitspecaccum uses predict.nls, and you can pass all arguments to that function. In addition, fitted, residuals, nobs, coef, AIC, logLik and deviance work on the result object.

Function specslope evaluates the derivative of the species accumulation curve at given number of sample plots, and gives the rate of increase in the number of species. The function works with specaccum result object when this is based on analytic models "exact", "rarefaction" or "coleman", and with non-linear regression results of fitspecaccum.

Nonlinear regression may fail for any reason, and some of the fitspecaccum models are fragile and may not succeed.

Value

Function specaccum returns an object of class "specaccum", and fitspecaccum a model of class "fitspecaccum" that adds a few items to the "specaccum" (see the end of the list below):

call	Function call.
method	Accumulator method.
sites	Number of sites. For method = "rarefaction" this is the number of sites corresponding to a certain number of individuals and generally not an integer, and the average number of individuals is also returned in item individuals.
effort	Average sum of weights corresponding to the number of sites when model was fitted with argument w
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.
weights	Matrix of accumulated weights corresponding to the columns of the perm matrix when model was fitted with argument w.

`fitted`, `residuals`, `coefficients`
: Only in `fitspecacum`: fitted values, residuals and nonlinear model coefficients. For `method = "random"` these are matrices with a column for each random accumulation.

`models`
: Only in `fitspecaccum`: list of fitted `nls` models (see Examples on accessing these models).

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

Chiarucci, A., Bacaro, G., Rocchini, D. & Fattorini, L. (2008). Discovering and rediscovering the sample-based rarefaction formula in the ecological literature. *Commun. Ecol.* 9: 121–123.

Coleman, B.D, Mares, M.A., Willis, M.R. & Hsieh, Y. (1982). Randomness, area and species richness. *Ecology* 63: 1121–1133.

Colwell, R.K., Chao, A., Gotelli, N.J., Lin, S.Y., Mao, C.X., Chazdon, R.L. & Longino, J.T. (2012). Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages. *J. Plant Ecol.* 5: 3–21.

Dengler, J. (2009). Which function describes the species-area relationship best? A review and empirical evaluation. *Journal of Biogeography* 36, 728–744.

Gotelli, N.J. & Colwell, R.K. (2001). Quantifying biodiversity: procedures and pitfalls in measurement and comparison of species richness. *Ecol. Lett.* 4, 379–391.

See Also

`rarefy` and `rrarefy` are related individual based models. Other accumulation models are `poolaccum` for extrapolated richness, and `renyiaccum` and `tsallisaccum` for diversity indices. 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="+")
## Fit Lomolino model to the exact accumulation
mod1 <- fitspecaccum(sp1, "lomolino")
```

```
coef(mod1)
fitted(mod1)
plot(sp1)
## Add Lomolino model using argument 'add'
plot(mod1, add = TRUE, col=2, lwd=2)
## Fit Arrhenius models to all random accumulations
mods <- fitspecaccum(sp2, "arrh")
plot(mods, col="hotpink")
boxplot(sp2, col = "yellow", border = "blue", lty=1, cex=0.3, add= TRUE)
## Use nls() methods to the list of models
sapply(mods$models, AIC)
```

---

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, smallsample = TRUE)
estimateR(x, ...)
specpool2vect(X, index = c("jack1","jack2", "chao", "boot","Species"))
poolaccum(x, permutations = 100, minsize = 3)
estaccumR(x, permutations = 100, parallel = getOption("mc.cores"))
## S3 method for class 'poolaccum'
summary(object, display, alpha = 0.05, ...)
## S3 method for class 'poolaccum'
plot(x, ...)
```

Arguments

`x`	Data frame or matrix with species data or the analysis result for the `plot` function.
`pool`	A vector giving a classification for pooling the sites in the species data. If missing, all sites are pooled together.
`smallsample`	Use small sample correction $(N-1)/N$, where N is the number of sites within the `pool`.
`X`, `object`	A `specpool` result object.
`index`	The selected index of extrapolated richness.
`permutations`	Usually an integer giving the number permutations, but can also be a list of control values for the permutations as returned by the function `how`, or a permutation matrix where each row gives the permuted indices.

`minsize`	Smallest number of sampling units reported.
`parallel`	Number of parallel processes or a predefined socket cluster. With `parallel = 1` uses ordinary, non-parallel processing. The parallel processing is done with **parallel** package.
`display`	Indices to be displayed.
`alpha`	Level of quantiles shown. This proportion will be left outside symmetric limits.
`...`	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	$S_P = S_0 + \frac{a_1^2}{2a_2}\frac{N-1}{N}$
Chao bias-corrected	$S_P = S_0 + \frac{a_1(a_1-1)}{2(a_2+1)}\frac{N-1}{N}$
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	$S_P = S_0 + \sum_{i=1}^{S_0}(1-p_i)^N$

`specpool` normally uses basic Chao equation, but when there are no doubletons ($a2 = 0$) it switches to bias-corrected version. In that case the Chao equation simplifies to $S_0 + \frac{1}{2}a_1(a_1-1)\frac{N-1}{N}$.

The abundance-based estimates in `estimateR` use counts (numbers of individuals) 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 bias-corrected Chao and ACE (O'Hara 2005, Chiu et al. 2014). The Chao estimate is similar as the bias corrected one above, but a_i refers to the number of species with abundance i instead of number of sites, and the small-sample correction is not used. The ACE estimate is defined as:

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

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 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 and are discussed in the vignette that can be read with the browseVignettes("vegan"). The standard error are based on the following sources: Chiu et al. (2014) for the Chao estimates and Smith and van Belle (1984) for the first-order Jackknife and the bootstrap (second-order jackknife is still missing). For the variance estimator of S_{ace} see O'Hara (2005).

Functions poolaccum and estaccumR are similar to specaccum, but estimate extrapolated richness indices of specpool or estimateR in addition to number of species for random ordering of sampling units. Function specpool uses presence data and estaccumR count data. The functions share summary and plot methods. The summary returns quantile envelopes of permutations corresponding the given level of alpha and standard deviation of permutations for each sample size. NB., these are not based on standard deviations estimated within specpool or estimateR, but they are based on permutations.

The plot function for accumulation models works both with poolaccum and estaccumR. It draws the mean of accumulations against the number of sites. **ggvegan** provides function autoplot that can also show the empirical confidence intervals of accumulations.

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. Functions poolaccum and estimateR return matrices of permutation results for each richness estimator, the vector of sample sizes and a table of means of permutations for each estimator.

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. In general, the functions give only the lower limit of species richness: the real richness is $S >= S_P$, and there is a consistent bias in the estimates. Even the bias-correction in Chao only reduces the bias, but does not remove it completely (Chiu et al. 2014).

Optional small sample correction was added to specpool in **vegan** 2.2-0. It was not used in the older literature (Chao 1987), but it is recommended recently (Chiu et al. 2014).

Author(s)

Bob O'Hara (estimateR) and Jari Oksanen.

References

Chao, A. (1987). Estimating the population size for capture-recapture data with unequal catchability. *Biometrics* 43, 783–791.

Chiu, C.H., Wang, Y.T., Walther, B.A. & Chao, A. (2014). Improved nonparametric lower bound of species richness via a modified Good-Turing frequency formula. *Biometrics* 70, 671–682.

Colwell, R.K. & Coddington, J.A. (1994). Estimating terrestrial biodiversity through extrapolation. *Phil. Trans. Roy. Soc. London* B 345, 101–118.

O'Hara, R.B. (2005). Species richness estimators: how many species can dance on the head of a pin? *J. Anim. Ecol.* 74, 375–386.

Palmer, M.W. (1990). The estimation of species richness by extrapolation. *Ecology* 71, 1195–1198.

Smith, E.P & van Belle, G. (1984). Nonparametric estimation of species richness. *Biometrics* 40, 119–129.

See Also

veiledspec, diversity, beals, specaccum.

Examples

```
data(dune)
data(dune.env)
pool <- with(dune.env, specpool(dune, Management))
pool
op <- par(mfrow=c(1,2))
boxplot(specnumber(dune) ~ Management, data = dune.env,
        col = "hotpink", border = "cyan3")
boxplot(specnumber(dune)/specpool2vect(pool) ~ Management,
        data = dune.env, col = "hotpink", border = "cyan3")
par(op)
data(BCI)
## Accumulation model
pool <- poolaccum(BCI)
summary(pool, display = "chao")
plot(pool)  ## use ggvegan::autoplot to show CI envelope
## Quantitative model
estimateR(BCI[1:5,])
```

---

sppscores *Add or Replace Species Scores in Distance-Based Ordination*

Description

Distance-based ordination (dbrda, capscale, metaMDS, monoMDS, wcmdscale) has no information on species, but some methods may add species scores if community data were available. However, the species scores may be missing (and they always are in dbrda and wcmdscale), or they may not have a close relation to used dissimilarity index. This function will add the species scores or replace the existing species scores in distance-based methods.

Usage

```
sppscores(object) <- value
```

Arguments

object	Ordination result from capscale, dbrda, metaMDS, monoMDS, pco or wcmdscale.
value	Community data to find the species scores.

Details

Distances have no information on species (columns, variables), and hence distance-based ordination has no information on species scores. However, the species scores can be added as supplementary information after the analysis to help the interpretation of results. Some ordination methods (capscale, metaMDS) can supplement the species scores during the analysis if community data were available in the analysis.

In capscale the species scores are found by projecting the community data to site ordination (linear combination scores), and the scores are accurate if the analysis used Euclidean distances. If the dissimilarity index can be expressed as Euclidean distances of transformed data (for instance, Chord and Hellinger Distances), the species scores based on transformed data will be accurate, but the function still finds the dissimilarities with untransformed data. Usually community dissimilarities differ in two significant ways from Euclidean distances: They are bound to maximum 1, and they use absolute differences instead of squared differences. In such cases, it may be better to use species scores that are transformed so that their Euclidean distances have a good linear relation to used dissimilarities. It is often useful to standardize data so that each row has unit total, and perform squareroot transformation to damp down the effect of squared differences (see Examples).

Functions dbrda and wcmdscale never find the species scores, but they mathematically similar to capscale, and similar rules should be followed when supplementing the species scores.

Functions for species scores in metaMDS and monoMDS use weighted averages (wascores) to find the species scores. These have better relationship with most dissimilarities than the projection scores used in metric ordination, but similar transformation of the community data should be used both in dissimilarities and in species scores.

Value

Replacement function adds the species scores or replaces the old scores in the ordination object.

Author(s)

Jari Oksanen

See Also

Function envfit finds similar scores, but based on correlations. The species scores for non-metric ordination use wascores which can also used directly on any ordination result.

Examples

```
data(BCI, BCI.env)
mod <- dbrda(vegdist(BCI) ~ Habitat, BCI.env)
## add species scores
sppscores(mod) <- BCI
## Euclidean distances of BCI differ from used dissimilarity
plot(vegdist(BCI), dist(BCI))
## more linear relationship
plot(vegdist(BCI), dist(decostand(BCI, "hellinger")))
## better species scores
sppscores(mod) <- decostand(BCI, "hellinger")
```

SSarrhenius *Self-Starting nls Species-Area Models*

Description

These functions provide self-starting species-area models for non-linear regression (nls). They can also be used for fitting species accumulation models in fitspecaccum. These models (and many more) are reviewed by Dengler (2009).

Usage

```
SSarrhenius(area, k, z)
SSgleason(area, k, slope)
SSgitay(area, k, slope)
SSlomolino(area, Asym, xmid, slope)
```

Arguments

area — Area or size of the sample: the independent variable.

k, z, slope, Asym, xmid — Estimated model parameters: see Details.

Details

These functions are intended to be used as self-starting models in non-linear regression (nls). There are several functions that can be used to further handle nls result, including summary, confint and many more (see nls).

All these functions are assumed to be used for species richness (number of species) as the independent variable, and area or sample size as the independent variable. Basically, these define least squares models of untransformed data, and will differ from models for transformed species richness or models with non-Gaussian error.

The Arrhenius model (SSarrhenius) is the expression k*area^z. This is the most classical model that can be found in any textbook of ecology (and also in Dengler 2009). Parameter z is the steepness of the species-area curve, and k is the expected number of species in a unit area.

The Gleason model (SSgleason) is a linear expression k + slope*log(area) (Dengler 200). This is a linear model, and starting values give the final estimates; it is provided to ease comparison with other models.

The Gitay model (SSgitay) is a quadratic logarithmic expression (k + slope*log(area))^2 (Gitay et al. 1991, Dengler 2009). Parameter slope is the steepness of the species-area curve, and k is the square root of expected richness in a unit area.

The Lomolino model (SSlomolino) is Asym/(1 + slope^log(xmid/area)) (Lomolino 2000, Dengler 2009). Parameter Asym is the asymptotic maximum number of species, slope is the maximum slope of increase of richness, and xmid is the area where half of the maximum richness is achieved.

In addition to these models, several other models studied by Dengler (2009) are available in standard R self-starting models: Michaelis-Menten (SSmicmen), Gompertz (SSgompertz), logistic (SSlogis), Weibull (SSweibull), and some others that may be useful.

Value

Numeric vector of the same length as `area`. It is the value of the expression of each model. If all arguments are names of objects the gradient matrix with respect to these names is attached as an attribute named `gradient`. This result object will be used in non-linear regression in function `nls` which returns its own result object with many support functions documented with `nls`.

Author(s)

Jari Oksanen.

References

Dengler, J. (2009) Which function describes the species-area relationship best? A review and empirical evaluation. *Journal of Biogeography* 36, 728–744.

Gitay, H., Roxburgh, S.H. & Wilson, J.B. (1991) Species-area relationship in a New Zealand tussock grassland, with implications for nature reserve design and for community structure. *Journal of Vegetation Science* 2, 113–118.

Lomolino, M. V. (2000) Ecology's most general, yet protean pattern: the species-area relationship. *Journal of Biogeography* 27, 17–26.

See Also

`nls`, `fitspecaccum`.

Examples

```
## Get species area data: sipoo.map gives the areas of islands
data(sipoo, sipoo.map)
S <- specnumber(sipoo)
plot(S ~ area, sipoo.map,  xlab = "Island Area (ha)",
  ylab = "Number of Species", ylim = c(1, max(S)))
## The Arrhenius model
marr <- nls(S ~ SSarrhenius(area, k, z), data=sipoo.map)
marr
summary(marr)
## confidence limits from profile likelihood
confint(marr)
## draw a line
xtmp <- with(sipoo.map, seq(min(area), max(area), len=51))
lines(xtmp, predict(marr, newdata=data.frame(area = xtmp)), lwd=2)
## The normal way is to use linear regression on log-log data,
## but this will be different from the previous:
mloglog <- lm(log(S) ~ log(area), data=sipoo.map)
summary(mloglog) # (Intercept) is log(k) of SSarrhenius result
lines(xtmp, exp(predict(mloglog, newdata=data.frame(area=xtmp))),
   lty=2)
## Gleason: log-linear
mgle <- nls(S ~ SSgleason(area, k, slope), sipoo.map)
lines(xtmp, predict(mgle, newdata=data.frame(area=xtmp)),
  lwd=2, col=2)
```

```
## Gitay: quadratic of log-linear
mgit <- nls(S ~ SSgitay(area, k, slope), sipoo.map)
lines(xtmp, predict(mgit, newdata=data.frame(area=xtmp)),
  lwd=2, col = 3)
## Lomolino: using original names of the parameters (Lomolino 2000):
mlom <- nls(S ~ SSlomolino(area, Smax, A50, Hill), sipoo.map)
summary(mlom)
lines(xtmp, predict(mlom, newdata=data.frame(area=xtmp)),
  lwd=2, col = 4)
## One canned model of standard R:
mmic <- nls(S ~ SSmicmen(area, Asym, slope), sipoo.map)
lines(xtmp, predict(mmic, newdata = data.frame(area=xtmp)),
  lwd =2, col = 5)
legend("bottomright", c("Arrhenius", "log-log linear", "Gleason", "Gitay",
  "Lomolino", "Michaelis-Menten"), col=c(1,1,2,3,4,5), lwd=c(2,1,2,2,2,2),
   lty=c(1,2,1,1,1,1))
## compare models (AIC)
allmods <- list(Arrhenius = marr, Gleason = mgle, Gitay = mgit,
   Lomolino = mlom, MicMen= mmic)
sapply(allmods, AIC)
```

---

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 semi-metric which means that they do not obey the triangle inequality $d_{ij} \leq d_{ik} + d_{kj}$, and shortest path algorithm can replace these dissimilarities as well, even when they are shorter than `toolong`.

De'ath (1999) suggested a simplified method known as extended dissimilarities, which are calculated with `path = "extended"`. In this method, dissimilarities that are `toolong` or longer are first made `NA`, and then the function tries to replace these `NA` dissimilarities with a path through single stepping stone points. If not all `NA` could be replaced with one pass, the function will make new passes with updated dissimilarities as long as all `NA` are replaced with extended dissimilarities. This mean that in the second and further passes, the remaining `NA` dissimilarities are allowed to have more than one stepping stone site, but previously replaced dissimilarities are not updated. Further, the function does not consider dissimilarities shorter than `toolong`, although some of these could be replaced with a shorter path in semi-metric indices, and used as a part of other paths. In optimal cases, the extended dissimilarities are equal to shortest paths, but they may be longer.

As an alternative to defining too long dissimilarities with parameter `toolong`, the input dissimilarities can contain `NA`s. If `toolong` is zero or negative, the function does not make any dissimilarities into `NA`. If there are no `NA`s in the input and `toolong = 0`, `path = "shortest"` will find shorter paths for semi-metric indices, and `path = "extended"` will do nothing. Function `no.shared` can be used to set dissimilarities to `NA`.

If the data are disconnected or there is no path between all points, the result will contain `NA`s and a warning is issued. Several methods cannot handle `NA` dissimilarities, and this warning should be taken seriously. Function `distconnected` can be used to find connected groups and remove rare outlier observations or groups of observations.

Alternative `path = "shortest"` uses Dijkstra's method for finding flexible shortest paths, implemented as priority-first search for dense graphs (Sedgewick 1990). Alternative `path = "extended"` follows De'ath (1999), but implementation is simpler than in his code.

Value

Function returns an object of class `dist` with extended dissimilarities (see functions `vegdist` and `dist`). The value of `path` is appended to the `method` attribute.

Note

The function changes the original dissimilarities, and not all like this. It may be best to use the function only when you really *must*: extremely high beta diversity where a large proportion of dissimilarities are at their upper limit (no species in common).

Semi-metric indices vary in their degree of violating the triangle inequality. Morisita and Horn–Morisita indices of `vegdist` may be very strongly semi-metric, and shortest paths can change these indices very much. Mountford index violates basic rules of dissimilarities: non-identical sites have zero dissimilarity if species composition of the poorer site is a subset of the richer. With Mountford index, you can find three sites i, j, k so that $d_{ik} = 0$ and $d_{jk} = 0$, but $d_{ij} > 0$. The results of `stepacross` on Mountford index can be very weird. If `stepacross` is needed, it is best to try to use it with more metric indices only.

Author(s)

Jari Oksanen

References

Bradfield, G.E. & Kenkel, N.C. (1987). Nonlinear ordination using flexible shortest path adjustment of ecological distances. *Ecology* 68, 750–753.

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

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

Williamson, M.H. (1978). The ordination of incidence data. *J. Ecol.* 66, 911-920.

See Also

Function `distconnected` can find connected groups in disconnected data, and function `no.shared` can be used to set dissimilarities as `NA`. See `swan` for an alternative approach. Function `stepacross` is an essential component in `isomap` and `cophenetic.spantree`.

Examples

```
# There are no data sets with high beta diversity in vegan, but this
# should give an idea.
data(dune)
dis <- vegdist(dune)
edis <- stepacross(dis)
plot(edis, dis, xlab = "Shortest path", ylab = "Original")
## Manhattan distance have no fixed upper limit.
dis <- vegdist(dune, "manhattan")
is.na(dis) <- no.shared(dune)
dis <- stepacross(dis, toolong=0)
```

---

stressplot.wcmdscale — *Display Ordination Distances Against Observed Distances in Eigenvector Ordinations*

Description

Functions plot ordination distances in given number of dimensions against observed distances or distances in full space in eigenvector methods. The display is similar as the Shepard diagram (`stressplot` for non-metric multidimensional scaling with `metaMDS` or `monoMDS`), but shows the linear relationship of the eigenvector ordinations. The `stressplot` methods are available for `wcmdscale`, `rda`, `cca`, `capscale`, `dbrda`, `prcomp` and `princomp`.

Usage

```
## S3 method for class 'wcmdscale'
stressplot(object, k = 2, pch, p.col = "blue", l.col = "red",
    lwd = 2, ...)
```

Arguments

`object`	Result object from eigenvector ordination (`wcmdscale`, `rda`, `cca`, `dbrda`, `capscale`)
`k`	Number of dimensions for which the ordination distances are displayed.
`pch`, `p.col`, `l.col`, `lwd`	Plotting character, point colour and line colour like in default `stressplot`
`...`	Other parameters to functions, e.g. graphical parameters.

Details

The functions offer a similar display for eigenvector ordinations as the standard Shepard diagram (`stressplot`) in non-metric multidimensional scaling. The ordination distances in given number of dimensions are plotted against observed distances. With metric distances, the ordination distances in full space (with all ordination axes) are equal to observed distances, and the fit line shows this equality. In general, the fit line does not go through the points, but the points for observed distances approach the fit line from below. However, with non-Euclidean distances (in `wcmdscale`, `dbrda` or `capscale`) with negative eigenvalues the ordination distances can exceed the observed distances in real dimensions; the imaginary dimensions with negative eigenvalues will correct these excess distances. If you have used `dbrda`, `capscale` or `wcmdscale` with argument `add` to avoid negative eigenvalues, the ordination distances will exceed the observed dissimilarities.

In partial ordination (`cca`, `rda`, and `capscale` with `Condition` in the formula), the distances in the partial component are included both in the observed distances and in ordination distances. With `k=0`, the ordination distances refer to the partial ordination. The exception is `dbrda` where the distances in partial, constrained and residual components are not additive, and only the first of these components can be shown, and partial models cannot be shown at all.

Value

Functions draw a graph and return invisibly the ordination distances or the ordination distances.

Author(s)

Jari Oksanen.

See Also

`stressplot` and `stressplot.monoMDS` for standard Shepard diagrams.

Examples

```
data(dune, dune.env)
mod <- rda(dune)
stressplot(mod)
mod <- rda(dune ~ Management, dune.env)
stressplot(mod, k=3)
```

taxondive *Indices of Taxonomic Diversity and Distinctness*

Description

Function finds indices of taxonomic diversity and distinctness, which are averaged taxonomic distances among species or individuals in the community (Clarke & Warwick 1998, 2001)

Usage

```
taxondive(comm, dis, match.force = FALSE)
taxa2dist(x, varstep = FALSE, check = TRUE, labels)
```

Arguments

comm	Community data.
dis	Taxonomic distances among taxa in comm. This should be a dist object or a symmetric square matrix.
match.force	Force matching of column names in comm and labels in dis. If FALSE, matching only happens when dimensions differ, and in that case the species must be in identical order in both.
x	Classification table with a row for each species or other basic taxon, and columns for identifiers of its classification at higher levels.
varstep	Vary step lengths between successive levels relative to proportional loss of the number of distinct classes.
check	If TRUE, remove all redundant levels which are different for all rows or constant for all rows and regard each row as a different basal taxon (species). If FALSE all levels are retained and basal taxa (species) also must be coded as variables (columns). You will get a warning if species are not coded, but you can ignore this if that was your intention.
labels	The labels attribute of taxonomic distances. Row names will be used if this is not given. Species will be matched by these labels in comm and dis in taxondive if these have different dimensions.

Details

Clarke & Warwick (1998, 2001) suggested several alternative indices of taxonomic diversity or distinctness. Two basic indices are called taxonomic diversity (Δ) and distinctness (Δ^*):

$$\Delta = (\sum\sum_{i<j} \omega_{ij} x_i x_j)/(n(n-1)/2)$$
$$\Delta^* = (\sum\sum_{i<j} \omega_{ij} x_i x_j)/(\sum\sum_{i<j} x_i x_j)$$

The equations give the index value for a single site, and summation goes over species i and j. Here ω are taxonomic distances among taxa, and x are species abundances, and n is the total abundance for a site. With presence/absence data both indices reduce to the same index Δ^+, and for this index

Clarke & Warwick (1998) also have an estimate of its standard deviation. Clarke & Warwick (2001) presented two new indices: $s\Delta^+$ is the product of species richness and Δ^+, and index of variation in taxonomic distinctness (Λ^+) defined as

$$\Lambda^+ = (\sum\sum_{i<j}\omega_{ij}^2)/(n(n-1)/2) - (\Delta^+)^2$$

The `dis` argument must be species dissimilarities. These must be similar to dissimilarities produced by `dist`. It is customary to have integer steps of taxonomic hierarchies, but other kind of dissimilarities can be used, such as those from phylogenetic trees or genetic differences. Further, the `dis` need not be taxonomic, but other species classifications can be used.

Function `taxa2dist` can produce a suitable `dist` object from a classification table. Each species (or basic taxon) corresponds to a row of the classification table, and columns give the classification at different levels. With `varstep = FALSE` the successive levels will be separated by equal steps, and with `varstep = TRUE` the step length is relative to the proportional decrease in the number of classes (Clarke & Warwick 1999). With `check = TRUE`, the function removes classes which are distinct for all species or which combine all species into one class, and assumes that each row presents a distinct basic taxon. The function scales the distances so that longest path length between taxa is 100 (not necessarily when `check = FALSE`).

Function `plot.taxondive` plots Δ^+ against Number of species, together with expectation and its approximate 2*sd limits. Function `summary.taxondive` finds the z values and their significances from Normal distribution for Δ^+.

Value

Function returns an object of class `taxondive` with following items:

`Species` Number of species for each site.

`D`, `Dstar`, `Dplus`, `SDplus`, `Lambda`
: Δ, Δ^*, Δ^+, $s\Delta^+$ and Λ^+ for each site.

`sd.Dplus` Standard deviation of Δ^+.

`ED`, `EDstar`, `EDplus`
: Expected values of corresponding statistics.

Function `taxa2dist` returns an object of class `"dist"`, with an attribute `"steps"` for the step lengths between successive levels.

Note

The function is still preliminary and may change. The scaling of taxonomic dissimilarities influences the results. If you multiply taxonomic distances (or step lengths) by a constant, the values of all Deltas will be multiplied with the same constant, and the value of Λ^+ by the square of the constant.

Author(s)

Jari Oksanen

References

Clarke, K.R & Warwick, R.M. (1998) A taxonomic distinctness index and its statistical properties. *Journal of Applied Ecology* 35, 523–531.

Clarke, K.R. & Warwick, R.M. (1999) The taxonomic distinctness measure of biodiversity: weighting of step lengths between hierarchical levels. *Marine Ecology Progress Series* 184: 21–29.

Clarke, K.R. & Warwick, R.M. (2001) A further biodiversity index applicable to species lists: variation in taxonomic distinctness. *Marine Ecology Progress Series* 216, 265–278.

See Also

diversity.

Examples

```
## Preliminary: needs better data and some support functions
data(dune)
data(dune.taxon)
# Taxonomic distances from a classification table with variable step lengths.
taxdis <- taxa2dist(dune.taxon, varstep=TRUE)
plot(hclust(taxdis), hang = -1)
# Indices
mod <- taxondive(dune, taxdis)
mod
summary(mod)
plot(mod)
```

---

tolerance *Species tolerances and sample heterogeneities.*

Description

Species tolerances and sample heterogeneities.

Usage

```
tolerance(x, ...)

## S3 method for class 'cca'
tolerance(x, choices = 1:2, which = c("species","sites"),
          scaling = "species", useN2 = TRUE, hill = FALSE, ...)

## S3 method for class 'decorana'
tolerance(x, data, choices = 1:4,
          which = c("sites", "species"), useN2 = TRUE, ...)
```

Arguments

x	object of class "cca".
choices	numeric; which ordination axes to compute tolerances and heterogeneities for. Defaults to axes 1 and 2.
which	character; one of "species" or "sites", indicating whether species tolerances or sample heterogeneities respectively are computed.
scaling	character or numeric; the ordination scaling to use. See scores.cca for details.
hill	logical; if scaling is a character, these control whether Hill's scaling is used for (C)CA respectively. See scores.cca for details.
useN2	logical; should the bias in the tolerances / heterogeneities be reduced via scaling by Hill's N2?
data	Original input data used in decorana. If missing, the function tries to get the same data as used in decorana call.
...	arguments passed to other methods.

Details

Species tolerance is a measure of response widths in the ordination space and it is estimated as a weighted dispersion of species scores to their site score. It describes the specialist vs. generalist responses of species. For sampling units it is a measure of weighted dispersion of species scores within a sampling unit and describes the heterogeneity of species composition in a sampling unit. Sum of squared tolerances is a measure of weighted within-species or within-site variation, while eigenvalue is a measure of between-species or between-site variation (see Examples).

Function implements Eq 6.47 and 6.48 from the Canoco 4.5 Reference Manual (pages 178–179).

Value

Matrix of tolerances/heterogeneities with some additional attributes: which, scaling, and N2, the latter of which will be NA if useN2 = FALSE or N2 could not be estimated.

Author(s)

Gavin L. Simpson and Jari Oksanen (decorana method).

Examples

```
data(dune, dune.env)
mod <- cca(dune ~ ., data = dune.env)

## defaults to species tolerances
tolerance(mod)

## sample heterogeneities for CCA axes 1:6
tolerance(mod, which = "sites", choices = 1:6)
## average should be 1 with scaling = "sites", hill = TRUE
tol <- tolerance(mod, which = "sites", scaling = "sites", hill = TRUE,
   choices = 1:4)
```

```
colMeans(tol)
apply(tol, 2, sd)
## Rescaling tries to set all tolerances to 1
tol <- tolerance(decorana(dune))
colMeans(tol)
apply(tol, 2, sd)

## Relation of tolerances (within-species variation) and eigenvalues
## (between-species variation) - with adequate 'scaling'.
tol <- tolerance(mod, what = "species", scaling = "sites", choices=1:4)
w <- weights(mod, "species")
all.equal(colSums(tol^2 * w),
          1 - eigenvals(mod)[1:4])
```

---

treedive *Functional Diversity and Community Distances from Species Trees*

Description

Functional diversity is defined as the total branch length in a trait dendrogram connecting all species, but excluding the unnecessary root segments of the tree (Petchey and Gaston 2006). Tree distance is the increase in total branch length when combining two sites.

Usage

```
treedive(comm, tree, match.force = TRUE, verbose = TRUE)
treeheight(tree)
treedist(x, tree, relative = TRUE, match.force = TRUE, ...)
```

Arguments

comm, x	Community data frame or matrix.
tree	A dendrogram which for treedive must be for species (columns).
match.force	Force matching of column names in data (comm, x) and labels in tree. If FALSE, matching only happens when dimensions differ (with a warning or message). The order of data must match to the order in tree if matching by names is not done.
verbose	Print diagnostic messages and warnings.
relative	Use distances relative to the height of combined tree.
...	Other arguments passed to functions (ignored).

Details

Function treeheight finds the sum of lengths of connecting segments in a dendrogram produced by hclust, or other dendrogram that can be coerced to a correct type using as.hclust. When applied to a clustering of species traits, this is a measure of functional diversity (Petchey and Gaston 2002, 2006), and when applied to phylogenetic trees this is phylogenetic diversity.

Function `treedive` finds the `treeheight` for each site (row) of a community matrix. The function uses a subset of dendrogram for those species that occur in each site, and excludes the tree root if that is not needed to connect the species (Petchey and Gaston 2006). The subset of the dendrogram is found by first calculating `cophenetic` distances from the input dendrogram, then reconstructing the dendrogram for the subset of the cophenetic distance matrix for species occurring in each site. Diversity is 0 for one species, and `NA` for empty communities.

Function `treedist` finds the dissimilarities among trees. Pairwise dissimilarity of two trees is found by combining species in a common tree and seeing how much of the tree height is shared and how much is unique. With `relative = FALSE` the dissimilarity is defined as $2(A \cup B) - A - B$, where A and B are heights of component trees and $A \cup B$ is the height of the combined tree. With `relative = TRUE` the dissimilarity is $(2(A \cup B) - A - B)/(A \cup B)$. Although the latter formula is similar to Jaccard dissimilarity (see `vegdist`, `designdist`), it is not in the range $0 \ldots 1$, since combined tree can add a new root. When two zero-height trees are combined into a tree of above zero height, the relative index attains its maximum value 2. The dissimilarity is zero from a combined zero-height tree.

The functions need a dendrogram of species traits or phylogenies as an input. If species traits contain `factor` or `ordered` factor variables, it is recommended to use Gower distances for mixed data (function `daisy` in package **cluster**), and usually the recommended clustering method is UPGMA (`method = "average"` in function `hclust`) (Podani and Schmera 2006). Phylogenetic trees can be changed into dendrograms using function `as.hclust.phylo` in the **ape** package.

It is possible to analyse the non-randomness of tree diversity using `oecosimu`. This needs specifying an adequate Null model, and the results will change with this choice.

Value

A vector of diversity values or a single tree height, or a dissimilarity structure that inherits from `dist` and can be used similarly.

Author(s)

Jari Oksanen

References

Lozupone, C. and Knight, R. 2005. UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology* 71, 8228–8235.

Petchey, O.L. and Gaston, K.J. 2002. Functional diversity (FD), species richness and community composition. *Ecology Letters* 5, 402–411.

Petchey, O.L. and Gaston, K.J. 2006. Functional diversity: back to basics and looking forward. *Ecology Letters* 9, 741–758.

Podani J. and Schmera, D. 2006. On dendrogram-based methods of functional diversity. *Oikos* 115, 179–185.

See Also

Function `treedive` is similar to the phylogenetic diversity function `pd` in the package **picante**, but excludes tree root if that is not needed to connect species. Function `treedist` is similar to

the phylogenetic similarity phylosor in the package **picante**, but excludes unneeded tree root and returns distances instead of similarities.

taxondive is something very similar from another bubble.

Examples

```
## There is no data set on species properties yet, and we demonstrate
## the methods using phylogenetic trees
data(dune)
data(dune.phylodis)
cl <- hclust(dune.phylodis)
treedive(dune, cl)
## Significance test using Null model communities.
## The current choice fixes numbers of species and picks species
## proportionally to their overall frequency
oecosimu(dune, treedive, "r1", tree = cl, verbose = FALSE)
## Phylogenetically ordered community table
dtree <- treedist(dune, cl)
tabasco(dune, hclust(dtree), cl)
## Use tree distances in distance-based RDA
dbrda(dtree ~ 1)
```

---

tsallis *Tsallis Diversity and Corresponding Accumulation Curves*

Description

Function tsallis find Tsallis diversities with any scale or the corresponding evenness measures. Function tsallisaccum finds these statistics with accumulating sites.

Usage

```
tsallis(x, scales = seq(0, 2, 0.2), norm = FALSE, hill = FALSE)
tsallisaccum(x, scales = seq(0, 2, 0.2), permutations = 100,
   raw = FALSE, subset, ...)
## S3 method for class 'tsallisaccum'
persp(x, theta = 220, phi = 15, col = heat.colors(100), zlim, ...)
```

Arguments

x	Community data matrix or plotting object.
scales	Scales of Tsallis diversity.
norm	Logical, if TRUE diversity values are normalized by their maximum (diversity value at equiprobability conditions).
hill	Calculate Hill numbers.
permutations	Usually an integer giving the number permutations, but can also be a list of control values for the permutations as returned by the function how, or a permutation matrix where each row gives the permuted indices.

raw	If FALSE then return summary statistics of permutations, and if TRUE then returns the individual permutations.
subset	logical expression indicating sites (rows) to keep: missing values are taken as FALSE.
theta, phi	angles defining the viewing direction. theta gives the azimuthal direction and phi the colatitude.
col	Colours used for surface.
zlim	Limits of vertical axis.
...	Other arguments which are passed to tsallis and to graphical functions.

Details

The Tsallis diversity (also equivalent to Patil and Taillie diversity) is a one-parametric generalised entropy function, defined as:

$$H_q = \frac{1}{q-1}(1 - \sum_{i=1}^{S} p_i^q)$$

where q is a scale parameter, S the number of species in the sample (Tsallis 1988, Tothmeresz 1995). This diversity is concave for all $q > 0$, but non-additive (Keylock 2005). For $q = 0$ it gives the number of species minus one, as q tends to 1 this gives Shannon diversity, for $q = 2$ this gives the Simpson index (see function diversity).

If norm = TRUE, tsallis gives values normalized by the maximum:

$$H_q(max) = \frac{S^{1-q} - 1}{1 - q}$$

where S is the number of species. As q tends to 1, maximum is defined as $ln(S)$.

If hill = TRUE, tsallis gives Hill numbers (numbers equivalents, see Jost 2007):

$$D_q = (1 - (q-1)H)^{1/(1-q)}$$

Details on plotting methods and accumulating values can be found on the help pages of the functions renyi and renyiaccum.

Value

Function tsallis returns a data frame of selected indices. Function tsallisaccum with argument raw = FALSE returns a three-dimensional array, where the first dimension are the accumulated sites, second dimension are the diversity scales, and third dimension are the summary statistics mean, stdev, min, max, Qnt 0.025 and Qnt 0.975. With argument raw = TRUE the statistics on the third dimension are replaced with individual permutation results.

Author(s)

Péter Sólymos, <solymos@ualberta.ca>, based on the code of Roeland Kindt and Jari Oksanen written for renyi

References

Tsallis, C. (1988) Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phis.* 52, 479–487.

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

Patil, G. P. and Taillie, C. (1982) Diversity as a concept and its measurement. *J. Am. Stat. Ass.* **77**, 548–567.

Keylock, C. J. (2005) Simpson diversity and the Shannon-Wiener index as special cases of a generalized entropy. *Oikos* **109**, 203–207.

Jost, L (2007) Partitioning diversity into independent alpha and beta components. *Ecology* **88**, 2427–2439.

See Also

Accumulation routines are based on functions `renyi` and `renyiaccum`, and can be plotted with functions `plot.renyi` and `plot.renyiaccum`. Alternative plots are provided by **ggvegan**. An object of class `tsallisaccum` can be displayed with dynamic 3D function `rgl.renyiaccum` in the **vegan3d** package. See also settings for `persp`.

Examples

```
data(BCI)
i <- sample(nrow(BCI), 12)
x1 <- tsallis(BCI[i,])
x1
plot(x1)
diversity(BCI[i,],"simpson") == x1[["2"]]
x2 <- tsallis(BCI[i,], norm=TRUE)
x2
plot(x2)
mod1 <- tsallisaccum(BCI)
persp(mod1)
plot(mod1)
mod2 <- tsallisaccum(BCI, norm=TRUE)
persp(mod2, theta=100, phi=30)
```

`varespec` *Vegetation and environment in lichen pastures*

Description

The `varespec` data frame has 24 rows and 44 columns. Columns are estimated cover values of 44 species. The variable names are formed from the scientific names, and are self explanatory for anybody familiar with the vegetation type. The `varechem` data frame has 24 rows and 14 columns, giving the soil characteristics of the very same sites as in the `varespec` data frame. The chemical measurements have obvious names. `Baresoil` gives the estimated cover of bare soil, `Humdepth` the thickness of the humus layer.

Usage

```
data(varechem)
data(varespec)
```

References

Väre, H., Ohtonen, R. and Oksanen, J. (1995) Effects of reindeer grazing on understorey vegetation in dry Pinus sylvestris forests. *Journal of Vegetation Science* 6, 523–530.

Examples

```
data(varespec)
data(varechem)
```

`varpart` *Partition the Variation of Community Matrix by 2, 3, or 4 Explanatory Matrices*

Description

The function partitions the variation in community data or community dissimilarities with respect to two, three, or four explanatory tables, using adjusted R^2 in redundancy analysis ordination (RDA) or distance-based redundancy analysis. If response is a single vector, partitioning is by partial regression. Collinear variables in the explanatory tables do NOT have to be removed prior to partitioning.

Usage

```
varpart(Y, X, ..., data, chisquare = FALSE, transfo, scale = FALSE,
    add = FALSE, sqrt.dist = FALSE, permutations)
## S3 method for class 'varpart'
summary(object, ...)
## S3 method for class 'varpart'
plot(x, Xnames = x$tables, ...)
## S3 method for class 'varpart234'
plot(x, cutoff = 0, digits = 1, ...)
showvarparts(parts, labels, bg = NULL, alpha = 0.25, Xnames,
    id.size = 1.2,  ...)
```

Arguments

`Y` Data frame or matrix containing the response data table or dissimilarity structure inheriting from `dist`. In community ecology, that table is often a site-by-species table or a dissimilarity object.

X	Two to four explanatory models, variables or tables. These can be defined in three alternative ways: (1) one-sided model formulae beginning with ~ and then defining the model, (2) name of a single numeric or factor variable, or (3) name of matrix with numeric or data frame with numeric and factor variables. The model formulae can have factors, interaction terms and transformations of variables. The names of the variables in the model formula are found in data frame given in `data` argument, and if not found there, in the user environment. Single variables, data frames or matrices are found in the user environment. All entries till the next argument (`data` or `transfo`) are interpreted as explanatory models, and the names of these extra arguments cannot be abbreviated nor omitted.
...	Other parameters passed to functions. NB, arguments after dots cannot be abbreviated but they must be spelt out completely. `plot` functions pass arguments `showvarparts`.
data	The data frame with the variables used in the formulae in `X`.
chisquare	Partition Chi-square or the inertia of Correspondence Analysis (`cca`).
transfo	Transformation for `Y` (community data) using `decostand`. All alternatives in `decostand` can be used, and those preserving Euclidean metric include `"hellinger"`, `"chi.square"`, `"total"`, `"norm"`. Ignored if `Y` are dissimilarities.
scale	Should the columns of `Y` be standardized to unit variance. Ignored if `Y` are dissimilarities.
add	Add a constant to the non-diagonal values to euclidify dissimilarities (see `wcmdscale` for details). Choice `"lingoes"` (or `TRUE`) use the recommended method of Legendre & Anderson (1999: "method 1") and `"cailliez"` uses their "method 2". The argument has an effect only when `Y` are dissimilarities.
sqrt.dist	Take square root of dissimilarities. This often euclidifies dissimilarities. NB., the argument name cannot be abbreviated. The argument has an effect only when `Y` are dissimilarities.
permutations	If `chisquare = TRUE`, the adjusted R^2 is estimated by permutations, and this parameter can be a list of control values for the permutations as returned by the function `how`, or the number of permutations required, or a permutation matrix where each row gives the permuted indices.
parts	Number of explanatory tables (circles) displayed.
labels	Labels used for displayed fractions. Default is to use the same letters as in the printed output.
bg	Fill colours of circles or ellipses.
alpha	Transparency of the fill colour. The argument takes precedence over possible transparency definitions of the colour. The value must be in range [0, 1], and low values are more transparent.
Xnames, id.size	Names for sources of variation. `Xnames=""` or `Xnames=NA` plot no names. If the specified names do not fit the figure space, names `X1` etc will be used. `id.size` is a numerical value for the character expansion of `Xnames`.
x, object	The `varpart` result.
cutoff	The values below `cutoff` will not be displayed.
digits	The number of significant digits; the number of decimal places is at least one higher.

Details

The functions partition the variation in Y into components accounted for by two to four explanatory tables and their combined effects. If Y is a multicolumn data frame or matrix, the partitioning is based on redundancy analysis (RDA, see rda) or on constrained correspondence analysis if chisquare = TRUE (CCA, see cca). If Y is a single variable, the partitioning is based on linear regression. If Y are dissimilarities, the decomposition is based on distance-based redundancy analysis (db-RDA, see dbrda) following McArdle & Anderson (2001). The input dissimilarities must be compatible to the results of dist. **Vegan** functions vegdist, designdist, raupcrick and betadiver produce such objects, as do many other dissimilarity functions in R packages. Partitioning will be made to squared dissimilarities analogously to using variance with rectangular data – unless sqrt.dist = TRUE was specified.

The function primarily uses adjusted R^2 to assess the partitions explained by the explanatory tables and their combinations (see RsquareAdj), because this is the only unbiased method (Peres-Neto et al., 2006). The raw R^2 for basic fractions are also displayed, but these are biased estimates of variation explained by the explanatory table. In correspondence analysis (chisquare = TRUE), the adjusted R^2 are found by permutation and they vary in repeated analyses.

The identifiable fractions are designated by lower case alphabets. The meaning of the symbols can be found in the separate document (use browseVignettes("vegan")), or can be displayed graphically using function showvarparts.

A fraction is testable if it can be directly expressed as an RDA or db-RDA model. In these cases the printed output also displays the corresponding RDA model using notation where explanatory tables after | are conditions (partialled out; see rda for details). Although single fractions can be testable, this does not mean that all fractions simultaneously can be tested, since the number of testable fractions is higher than the number of estimated models. The non-testable components are found as differences of testable components. The testable components have permutation variance in correspondence analysis (chisquare = TRUE), and the non-testable components have even higher variance.

An abridged explanation of the alphabetic symbols for the individual fractions follows, but computational details should be checked in the vignette (readable with browseVignettes("vegan")) or in the source code.

With two explanatory tables, the fractions explained uniquely by each of the two tables are [a] and [b], and their joint effect is [c].

With three explanatory tables, the fractions explained uniquely by each of the three tables are [a] to [c], joint fractions between two tables are [d] to [f], and the joint fraction between all three tables is [g].

With four explanatory tables, the fractions explained uniquely by each of the four tables are [a] to [d], joint fractions between two tables are [e] to [j], joint fractions between three variables are [k] to [n], and the joint fraction between all four tables is [o].

summary will give an overview of unique and and overall contribution of each group of variables. The overall contribution (labelled as "Contributed") consists of the unique contribution of the variable and equal shares of each fraction where the variable contributes. The summary tabulates how each fraction is divided between the variables, and the contributed component is the sum of all these divided fractions. The summary is based on the idea of Lai et al. (2022), and is similar to the output of their **rdacca.hp** package.

There is a plot function that displays the Venn diagram and labels each intersection (individual fraction) with the adjusted R squared if this is higher than cutoff. A helper function showvarpart

displays the fraction labels. The circles and ellipses are labelled by short default names or by names defined by the user in argument `Xnames`. Longer explanatory file names can be written on the varpart output plot as follows: use option `Xnames=NA`, then add new names using the `text` function. A bit of fiddling with coordinates (see `locator`) and character size should allow users to place names of reasonably short lengths on the `varpart` plot.

Value

Function `varpart` returns an object of class `"varpart"` with items `scale` and `transfo` (can be missing) which hold information on standardizations, `tables` which contains names of explanatory tables, and `call` with the function `call`. The function `varpart` calls function `varpart2`, `varpart3` or `varpart4` which return an object of class `"varpart234"` and saves its result in the item `part`. The items in this object are:

`SS.Y`	Sum of squares of matrix `Y`.
`n`	Number of observations (rows).
`nsets`	Number of explanatory tables
`bigwarning`	Warnings on collinearity.
`fract`	Basic fractions from all estimated constrained models.
`indfract`	Individual fractions or all possible subsections in the Venn diagram (see `showvarparts`).
`contr1`	Fractions that can be found after conditioning on single explanatory table in models with three or four explanatory tables.
`contr2`	Fractions that can be found after conditioning on two explanatory tables in models with four explanatory tables.

Fraction Data Frames

Items `fract`, `indfract`, `contr1` and `contr2` are all data frames with items:

`Df`: Degrees of freedom of numerator of the F-statistic for the fraction.

`R.square`: Raw R^2. This is calculated only for `fract` and this is `NA` in other items.

`Adj.R.square`: Adjusted R^2.

`Testable`: If the fraction can be expressed as a (partial) RDA model, it is directly `Testable`, and this field is `TRUE`. In that case the fraction label also gives the specification of the testable RDA model.

Note

You can use command `browseVignettes("vegan")` to display document which presents Venn diagrams showing the fraction names in partitioning the variation of Y with respect to 2, 3, and 4 tables of explanatory variables, as well as the equations used in variation partitioning.

The functions frequently give negative estimates of variation. Adjusted R^2 can be negative for any fraction; unadjusted R^2 of testable fractions of variances will be non-negative. Non-testable fractions cannot be found directly, but by subtracting different models, and these subtraction results can be negative. The fractions are orthogonal, or linearly independent, but more complicated or nonlinear dependencies can cause negative non-testable fractions. Any fraction can be negative for

non-Euclidean dissimilarities because the underlying db-RDA model can yield negative eigenvalues (see dbrda). These negative eigenvalues in the underlying analysis can be avoided with arguments sqrt.dist and add which have a similar effect as in dbrda: the square roots of several dissimilarities do not have negative eigenvalues, and no negative eigenvalues are produced after Lingoes or Cailliez adjustment, which in effect add random variation to the dissimilarities.

A simplified, fast version of RDA, CCA and dbRDA are used (functions simpleRDA2, simpleCCA and simpleDBRDA). The actual calculations are done in functions varpart2 to varpart4, but these are not intended to be called directly by the user.

Author(s)

Pierre Legendre, Departement de Sciences Biologiques, Universite de Montreal, Canada. Further developed by Jari Oksanen.

References

(a) References on variation partitioning

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

Lai J., Y. Zou, J. Zhang & P. Peres-Neto. 2022. Generalizing hierarchical and variation partitioning in multiple regression and canonical analysis using the rdacca.hp R package. Methods in Ecology and Evolution, 13: 782–788.

Legendre, P. & L. Legendre. 2012. Numerical ecology, 3rd English edition. Elsevier Science BV, Amsterdam.

(b) Reference on transformations for species data

Legendre, P. and E. D. Gallagher. 2001. Ecologically meaningful transformations for ordination of species data. Oecologia 129: 271–280.

(c) Reference on adjustment of the bimultivariate redundancy statistic

Peres-Neto, P., P. Legendre, S. Dray and D. Borcard. 2006. Variation partitioning of species data matrices: estimation and comparison of fractions. Ecology 87: 2614–2625.

(d) References on partitioning of dissimilarities

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

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

See Also

For analysing testable fractions, see rda and anova.cca. For data transformation, see decostand. Function inertcomp gives (unadjusted) components of variation for each species or site separately. Function rda displays unadjusted components in its output, but RsquareAdj will give adjusted R^2 that are similar to the current function also for partial models.

Examples

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

# Two explanatory data frames -- Hellinger-transform Y
mod <- varpart(mite, mite.env, mite.pcnm, transfo="hel")
mod
summary(mod)

## Use fill colours
showvarparts(2, bg = c("hotpink","skyblue"))
plot(mod, bg = c("hotpink","skyblue"))
## Test fraction [a] using partial RDA, '~ .' in formula tells to use
## all variables of data mite.env.
aFrac <- rda(decostand(mite, "hel"), mite.env, mite.pcnm)
anova(aFrac)
## RsquareAdj gives the same result as component [a] of varpart
RsquareAdj(aFrac)

## Partition Bray-Curtis dissimilarities
varpart(vegdist(mite), mite.env, mite.pcnm)
## Three explanatory tables with formula interface
mod <- varpart(mite, ~ SubsDens + WatrCont, ~ Substrate + Shrub + Topo,
   mite.pcnm, data=mite.env, transfo="hel")
mod
summary(mod)
showvarparts(3, bg=2:4)
plot(mod, bg=2:4)

## Use RDA to test fraction [a]
## Matrix can be an argument in formula
rda.result <- rda(decostand(mite, "hell") ~ SubsDens + WatrCont +
   Condition(Substrate + Shrub + Topo) +
   Condition(as.matrix(mite.pcnm)), data = mite.env)
anova(rda.result)

## Four explanatory tables
mod <- varpart(mite, ~ SubsDens + WatrCont, ~Substrate + Shrub + Topo,
  mite.pcnm[,1:11], mite.pcnm[,12:22], data=mite.env, transfo="hel")
mod
summary(mod)
plot(mod, bg=2:5)
## Show values for all partitions by putting 'cutoff' low enough:
plot(mod, cutoff = -Inf, cex = 0.7, bg=2:5)
```

`vegan-deprecated` *Deprecated Functions in vegan package*

Description

These functions are provided for compatibility with older versions of **vegan** only, and may be defunct as soon as the next release.

Details

None.

See Also

Deprecated

---

vegdist *Dissimilarity Indices for Community Ecologists*

Description

The function computes dissimilarity indices that are useful for or popular with community ecologists. All indices use quantitative data, although they would be named by the corresponding binary index, but you can calculate the binary index using an appropriate argument. If you do not find your favourite index here, you can see if it can be implemented using designdist. Gower, Bray–Curtis, Jaccard and Kulczynski indices are good in detecting underlying ecological gradients (Faith et al. 1987). Morisita, Horn–Morisita, Binomial, Cao and Chao indices should be able to handle different sample sizes (Wolda 1981, Krebs 1999, Anderson & Millar 2004), and Mountford (1962) and Raup-Crick indices for presence–absence data should be able to handle unknown (and variable) sample sizes. Most of these indices are discussed by Krebs (1999) and Legendre & Legendre (2012), and their properties further compared by Wolda (1981) and Legendre & De Cáceres (2012). Aitchison (1986) distance is equivalent to Euclidean distance between CLR-transformed samples ("clr") and deals with positive compositional data. Robust Aitchison distance by Martino et al. (2019) uses robust CLR ("rlcr"), making it applicable to non-negative data including zeroes (unlike the standard Aitchison).

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", "clark", "bray", "kulczynski", "jaccard", "gower", "altGower", "morisita", "horn", "mountford", "raup", "binomial", "chao", "cao", "mahalanobis", "chisq", "chord", "hellinger", "aitchison", or "robust.aitchison".
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 (but some dissimilarities may still be `NA`, although calculation is handled).
`...`	Other parameters. These are ignored, except in `method ="gower"` which accepts `range.global` parameter of `decostand`, and in `method="aitchison"`, which accepts `pseudocount` parameter of `decostand` used in the `clr` transformation.

Details

Jaccard (`"jaccard"`), Mountford (`"mountford"`), Raup–Crick (`"raup"`), Binomial and Chao indices are discussed later in this section. The function also finds indices for presence/ absence data by setting `binary = TRUE`. The following overview gives first the quantitative version, where x_{ij} x_{ik} refer to the quantity on species (column) i and sites (rows) j and k. In binary versions A and B are the numbers of species on compared sites, and J is the number of species that occur on both compared sites similarly as in `designdist` (many indices produce identical binary versions):

`euclidean`	$d_{jk} = \sqrt{\sum_i (x_{ij} - x_{ik})^2}$ binary: $\sqrt{A + B - 2J}$
`manhattan`	$d_{jk} = \sum_i \lvert x_{ij} - x_{ik} \rvert$ binary: $A + B - 2J$
`gower`	$d_{jk} = (1/M) \sum_i \frac{\lvert x_{ij} - x_{ik} \rvert}{\max x_i - \min x_i}$ binary: $(A + B - 2J)/M$ where M is the number of columns (excluding missing values)
`altGower`	$d_{jk} = (1/NZ) \sum_i \lvert x_{ij} - x_{ik} \rvert$ where NZ is the number of non-zero columns excluding double-zeros (Anderson et al. 2006). binary: $\frac{A+B-2J}{A+B-J}$
`canberra`	$d_{jk} = \frac{1}{NZ} \sum_i \frac{\lvert x_{ij} - x_{ik} \rvert}{\lvert x_{ij} \rvert + \lvert x_{ik} \rvert}$ where NZ is the number of non-zero entries. binary: $\frac{A+B-2J}{A+B-J}$
`clark`	$d_{jk} = \sqrt{\frac{1}{NZ} \sum_i (\frac{x_{ij} - x_{ik}}{x_{ij} + x_{ik}})^2}$ where NZ is the number of non-zero entries. binary: $\frac{A+B-2J}{A+B-J}$
`bray`	$d_{jk} = \frac{\sum_i \lvert x_{ij} - x_{ik} \rvert}{\sum_i (x_{ij} + x_{ik})}$ binary: $\frac{A+B-2J}{A+B}$
`kulczynski`	$d_{jk} = 1 - 0.5(\frac{\sum_i \min(x_{ij}, x_{ik})}{\sum_i x_{ij}} + \frac{\sum_i \min(x_{ij}, x_{ik})}{\sum_i x_{ik}})$ binary: $1 - (J/A + J/B)/2$
`morisita`	$d_{jk} = 1 - \frac{2 \sum_i x_{ij} x_{ik}}{(\lambda_j + \lambda_k) \sum_i x_{ij} \sum_i x_{ik}}$, where $\lambda_j = \frac{\sum_i x_{ij}(x_{ij} - 1)}{\sum_i x_{ij} \sum_i (x_{ij} - 1)}$ binary: cannot be calculated
`horn`	Like `morisita`, but $\lambda_j = \sum_i x_{ij}^2 / (\sum_i x_{ij})^2$ binary: $\frac{A+B-2J}{A+B}$
`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 $n_i = x_{ij} + x_{ik}$ binary: $\log(2) \times (A + B - 2J)$
`cao`	$d_{jk} = \frac{1}{S} \sum_i \log\left(\frac{n_i}{2}\right) - (x_{ij} \log(x_{ik}) + x_{ik} \log(x_{ij}))/n_i$,

where S is the number of species in compared sites and $n_i = x_{ij} + x_{ik}$

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).

Cao index or CYd index (Cao et al. 1997) was suggested as a minimally biased index for high beta diversity and variable sampling intensity. Cao index does not have a fixed upper limit, but can vary among sites with no shared species. The index is intended for count (integer) data, and it is undefined for zero abundances; these are replaced with arbitrary value 0.1 following Cao et al. (1997). Cao et al. (1997) used $\log_{10}$, but the current function uses natural logarithms so that the values are approximately 2.30 times higher than with 10-based logarithms. Anderson & Thompson (2004) give an alternative formulation of Cao index to highlight its relationship with Binomial index (above).

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 $0 \ldots \log(2)$.

Raup–Crick dissimilarity (`method = "raup"`) is a probabilistic index based on presence/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. The current function uses analytic result from hypergeometric distribution (`phyper`) to find the probabilities. 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. The index uses equal occurrence probabilities for all species, but Raup and Crick originally suggested that sampling probabilities should be proportional to species frequencies (Chase et al. 2011). A simulation approach with unequal species sampling probabilities is implemented in `raupcrick` function following Chase et al. (2011). The index can be also used for transposed data to give a probabilistic dissimilarity index of species co-occurrence (identical to Veech 2013).

Chao index tries to take into account the number of unseen species pairs, similarly as in `method = "chao"` in `specpool`. Function `vegdist` implements a Jaccard, index defined as $1 - \frac{U \times V}{U+V-U \times V}$; other types can be defined with function `chaodist`. In Chao equation, $U = C_j/N_j + (N_k - 1)/N_k \times a_1/(2a_2) \times S_j/N_j$, and V is similar except for site index k. C_j is the total number of individuals in the species of site j that are shared with site k, N_j is the total number of individuals at site j, a_1 (and a_2) are the number of species occurring in site j that have only one (or two) individuals in site

k, and S_j is the total number of individuals in the species present at site j that occur with only one individual in site k (Chao et al. 2005).

Morisita index can be only used with genuine count data (integers). It is based on the idea of resampling without replacement in finite samples and should not be used with presence/absence data, and gives meaningless results if compared sampling units (rows) have largest integer 1. Its Horn–Morisita variant is able to handle any abundance data, and should be used if data are unsuitable for Morisita.

Mahalanobis distances are Euclidean distances of a matrix where columns are centred, have unit variance, and are uncorrelated. The index is not commonly used for community data, but it is sometimes used for environmental variables. The calculation is based on transforming data matrix and then using Euclidean distances following Mardia et al. (1979). The Mahalanobis transformation usually fails when the number of columns is larger than the number of rows (sampling units). When the transformation fails, the distances are nearly constant except for small numeric noise. Users must check that the returned Mahalanobis distances are meaningful.

Euclidean and Manhattan dissimilarities are not good in gradient separation without proper standardization but are still included for comparison and special needs.

Chi-square distances (`"chisq"`) are Euclidean distances of Chi-square transformed data (see `decostand`). This is the internal standardization used in correspondence analysis (`cca`, `decorana`). Weighted principal coordinates analysis of these distances with row sums as weights is equal to correspondence analysis (see the Example in `wcmdscale`). Chi-square distance is intended for non-negative data, such as typical community data. However, it can be calculated as long as all margin sums are positive, but warning is issued on negative data entries.

Chord distances (`"chord"`) are Euclidean distance of a matrix where rows are standardized to unit norm (their sums of squares are 1) using `decostand`. Geometrically this standardization moves row points to a surface of multidimensional unit sphere, and distances are the chords across the hypersphere. Hellinger distances (`"hellinger"`) are related to Chord distances, but data are standardized to unit total (row sums are 1) using `decostand`, and then square root transformed. These distances have upper limit of $\sqrt{2}$.

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.

Aitchison distance (1986) and robust Aitchison distance (Martino et al. 2019) are metrics that deal with compositional data. Aitchison distance has been said to outperform Jensen-Shannon divergence and Bray-Curtis dissimilarity, due to a better stability to subsetting and aggregation, and it being a proper distance (Aitchison et al., 2000).

The naming conventions vary. The one adopted here is traditional rather than truthful to priority. The function finds either quantitative or binary variants of the indices under the same name, which correctly may refer only to one of these alternatives For instance, the Bray index is known also as Steinhaus, Czekanowski and Sørensen index. The quantitative version of Jaccard should probably called Ružička index. The abbreviation `"horn"` for the Horn–Morisita index is misleading, since there is a separate Horn index. The abbreviation will be changed if that index is implemented in `vegan`.

Value

Function is a drop-in replacement for dist function and returns a distance object of the same type. The result object adds attribute maxdist that gives the theoretical maximum of the index for sampling units that share no species, or NA when there is no such maximum.

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 implementation of Gower index that also can handle mixed data of numeric and class variables. There are two versions of Gower distance ("gower", "altGower") which differ in scaling: "gower" divides all distances by the number of observations (rows) and scales each column to unit range, but "altGower" omits double-zeros and divides by the number of pairs with at least one above-zero value, and does not scale columns (Anderson et al. 2006). You can use decostand to add range standardization to "altGower" (see Examples). Gower (1971) suggested omitting double zeros for presences, but it is often taken as the general feature of the Gower distances. See Examples for implementing the Anderson et al. (2006) variant of the Gower index.

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 composition without species and you should remove empty sites from community data.

Author(s)

Jari Oksanen, with contributions from Tyler Smith (Gower index), Michael Bedward (Raup–Crick index), and Leo Lahti (Aitchison and robust Aitchison distance).

References

Aitchison, J. The Statistical Analysis of Compositional Data (1986). London, UK: Chapman & Hall.

Aitchison, J., Barceló-Vidal, C., Martín-Fernández, J.A., Pawlowsky-Glahn, V. (2000). Logratio analysis and compositional distance. *Math. Geol.* **32**, 271–275.

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.

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

Anderson, M.J & Thompson, A.A. (2004). Multivariate control charts for ecological and environmental monitoring. *Ecological Applications* 14, 1921–1935.

Cao, Y., Williams, W.P. & Bark, A.W. (1997). Similarity measure bias in river benthic Auswuchs community analysis. *Water Environment Research* 69, 95–106.

Chao, A., Chazdon, R. L., Colwell, R. K. and Shen, T. (2005). A new statistical approach for assessing similarity of species composition with incidence and abundance data. *Ecology Letters* 8, 148–159.

Chase, J.M., Kraft, N.J.B., Smith, K.G., Vellend, M. and Inouye, B.D. (2011). Using null models to disentangle variation in community dissimilarity from variation in α-diversity. *Ecosphere* 2:art24 doi:10.1890/ES1000117.1

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

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics* 27, 623–637.

Krebs, C. J. (1999). *Ecological Methodology.* Addison Wesley Longman.

Legendre, P. & De Cáceres, M. (2012). Beta diversity as the variance of community data: dissimilarity coefficients and partitioning. *Ecology Letters* 16, 951–963. doi:10.1111/ele.12141

Legendre, P. and Legendre, L. (2012) *Numerical Ecology*. 3rd English ed. Elsevier.

Mardia, K.V., Kent, J.T. and Bibby, J.M. (1979). *Multivariate analysis*. Academic Press.

Martino, C., Morton, J.T., Marotz, C.A., Thompson, L.R., Tripathi, A., Knight, R. & Zengler, K. (2019) A novel sparse compositional technique reveals microbial perturbations. *mSystems* **4**, 1.

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.

Veech, J. A. (2013). A probabilistic model for analysing species co-occurrence. *Global Ecology and Biogeography* 22, 252–260.

Wolda, H. (1981). Similarity indices, sample size and diversity. *Oecologia* 50, 296–302.

See Also

Function `designdist` can be used for defining your own dissimilarity index. Function `betadiver` provides indices intended for the analysis of beta diversity.

Examples

```
data(varespec)
vare.dist <- vegdist(varespec)
# Orlóci's Chord distance: range 0 .. sqrt(2)
vare.dist <- vegdist(decostand(varespec, "norm"), "euclidean")
# Anderson et al.  (2006) version of Gower
vare.dist <- vegdist(decostand(varespec, "log"), "altGower")
# Range standardization with "altGower" (that excludes double-zeros)
vare.dist <- vegdist(decostand(varespec, "range"), "altGower")
# Robust Aitchison distance equals to Euclidean distance for rclr transformed data
vare.dist <- vegdist(decostand(varespec, "rclr"), method = "euclidean")
vare.dist <- vegdist(varespec, "robust.aitchison")
```

vegemite *Display Compact Ordered Community Tables*

Description

Functions vegemite and tabasco display compact community tables. Function vegemite prints text tables where species are rows, and each site takes only one column without spaces. Function tabasco provides interface for heatmap for a colour image of the data. The community table can be ordered by explicit indexing, by environmental variables or results from an ordination or cluster analysis.

Usage

```
vegemite(x, use, scale, sp.ind, site.ind, zero=".", select,
    diagonalize = FALSE, ...)
tabasco(x, use, sp.ind = NULL, site.ind = NULL, select,
    Rowv = TRUE, Colv = TRUE, labRow = NULL, labCol = NULL,
    scale, col = heat.colors(12), ...)
coverscale(x, scale=c("Braun.Blanquet", "Domin", "Hult", "Hill", "fix","log"),
           maxabund, character = TRUE)
```

Arguments

x	Community data.
use	Either a numeric vector or a classification factor, or an object from cca, decorana *etc.* or hclust or a dendrogram for ordering sites and species.
sp.ind, site.ind	Species and site indices. In tabasco, sp.ind can also be an hclust tree, agnes clustering or a dendrogram.
zero	Character used for zeros.
select	Select a subset of sites. This can be a logical vector (TRUE for selected sites), or a vector of indices of selected sites. The order of indices does not influence results, but you must specify use or site.ind to reorder sites.
diagonalize	Try to re-order vegemite table to a diagonal pattern when using classification factor or a dendrogram. Dendrograms are re-orded by the first axis of correspondence analysis. Factor levels and sites within factor levels are re-ordered by the first axis of (constrained) correspondence analysis.
Rowv, Colv	Re-order factors or dendrograms for the rows (sites) or columns (species) of x. If Rowv = TRUE, factor levels and sites within factors are re-ordered to show diagonal pattern using CCA. If Rowv = TRUE, row dendrograms are ordered by the first axis of correspondence analysis, and when Colv = TRUE column dendrograms by the weighted average (wascores) of the row order. Alternatively, the arguments can be vectors that are used to reorder the dendrogram.
labRow, labCol	character vectors with row and column labels used in the heatmap instead of the default. NB., the input matrix is transposed so that row labels will be used for data columns.

scale	In vegemite and coverscale: cover scale used (can be abbreviated). In tabasco: scaling of colours in heatmap. The alternatives of coverscale can be used in tabasco, and in addition "column" or "row" scale columns or rows to equal maxima (NB., these refer to the transposed data of the heatmap), while "none" uses original values.
col	A vector of colours used for above-zero abundance values.
maxabund	Maximum abundance used with scale = "log". Data maximum in the selected subset will be used if this is missing.
character	Return character codes suitable for tabulation in vegemite. If FALSE, returns corresponding integers suitable for numerical analysis.
...	Arguments passed to coverscale (i.e., maxabund) in vegemite and to heatmap in tabasco.

Details

The function vegemite prints a traditional community table. The display is transposed so that species are in rows and sites in columns. The table is printed in compact form: only one character can be used for abundance, and there are no spaces between columns. Species with no occurrences are dropped from the table.

Function tabasco produces a similar table as vegemite using heatmap, where abundances are coded by colours. The function scales the abundances to equal intervals for colour palette, but either rows or columns can be scaled to equal maxima, or a coverscale class systems can be used. The function can also display dendrograms for sites (columns) or species if these are given as an argument (use for sites, sp.ind for species).

The parameter use will be used to re-order output. The use can be a vector or an object from hclust or agnes, a dendrogram or any ordination result recognized by scores (all ordination methods in **vegan** and some of those not in **vegan**). The hclust, agnes and dendrogram must be for sites. The dendrogram is displayed above the sites in tabasco, but is not shown in vegemite. In tabasco the species dendrogram can be given in sp.ind.

If use is a numeric vector, it is used for ordering sites, and if it is a factor, it is used to order sites by classes. If use is an object from ordination, both sites and species are arranged by the first axis (provided that results are available also for species). When use is an object from hclust, agnes or dendrogram, the sites are ordered similarly as in the cluster dendrogram. Function tabasco re-orders the dendrogram to give a diagonal pattern if Rowv = TRUE. Alternatively, if Rowv is a vector its values are used to re-order dendrogram. With diagonalize = TRUE the dendrogram will be re-ordered in vegemite to give a diagonal pattern. If use is a factor, its levels and sites within levels will be reordered to give a diagonal pattern if diagonalize = TRUE in vegemite or Rowv = TRUE in tabasco. In all cases where species scores are missing, species are ordered by their weighted averages (wascores) on site order or site value.

Species and sites can be ordered explicitly giving their indices or names in parameters sp.ind and site.ind. Argument use takes precedence, and site.ind cannot be used together with use. However, sp.index takes precedence over species order of use. A subset of sites can be displayed using argument select, but this cannot be used to order sites, but you still must give use or site.ind. However, tabasco makes two exceptions: site.ind and select cannot be used when use is a dendrogram (clustering result). In addition, the sp.ind can be an hclust tree, agnes clustering or a dendrogram, and in that case the dendrogram is plotted on the left side of the heatmap, and Colv

can be used to reorder the dendrogram. Phylogenetic trees cannot be directly used, but package **ape** has tools to transform these to `hclust` trees.

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). Function `tabasco` can also use these traditional class scales, but it treats the transformed values as corresponding integers. Braun-Blanquet and Domin scales are actually not strict cover scales, and the limits used for codes `r` and `+` are arbitrary. Scale `Hill` may be inappropriately named, since Mark O. Hill probably never intended this as a cover scale. However, it is used as default "cut levels" in his `TWINSPAN`, and surprisingly many users stick to this default, and this is a *de facto* standard in publications. All traditional scales assume that values are cover percentages with maximum 100. However, non-traditional alternative `log` can be used with any scale range. Its class limits are integer powers of 1/2 of the maximum (argument `maxabund`), with `+` used for non-zero entries less than 1/512 of the maximum (`log` stands alternatively for logarithmic or logical). Scale `fix` is intended for "fixing" 10-point scales: it truncates scale values to integers, and replaces 10 with `X` and positive values below 1 with `+`.

Value

The functions are used mainly to display a table, but they return (invisibly) a list with items `species` for ordered species index, `sites` for ordered site index, and `vegemite` also `table` of the ordered community table.

These items can be used as arguments `sp.ind` and `site.ind` to reproduce the table, or the `table` can be further edited. In addition to the table, `vegemite` prints the numbers of species and sites and the name of the used cover scale.

Note

The name `vegemite` was chosen because the output is so compact, and the `tabasco` because it is just as compact, but hotter.

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' for `vegemite`. Function `tabasco` is based on `heatmap` which in turn is based on `image`. Both functions order species with weighted averages using `wascores`.

Examples

```
data(varespec)
## Print only more common species
freq <- apply(varespec > 0, 2, sum)
vegemite(varespec, scale="Hult", sp.ind = freq > 10)
## Order by correspondence analysis, use Hill scaling and layout:
dca <- decorana(varespec)
vegemite(varespec, dca, "Hill", zero="-")
## Show one class from cluster analysis, but retain the ordering above
clus <- hclust(vegdist(varespec))
cl <- cutree(clus, 3)
sel <- vegemite(varespec, use=dca, select = cl == 3, scale="Br")
## Re-create previous
vegemite(varespec, sp=sel$sp, site=sel$site, scale="Hult")
## Re-order clusters by ordination
clus <- as.dendrogram(clus)
clus <- reorder(clus, scores(dca, choices=1, display="sites"), agglo.FUN = mean)
vegemite(varespec, clus, scale = "Hult")

## Abundance values have such a wide range that they must be rescaled
tabasco(varespec, dca, scale="Braun")

## Classification trees for species
data(dune, dune.taxon)
taxontree <- hclust(taxa2dist(dune.taxon))
plotree <- hclust(vegdist(dune), "average")
## Automatic reordering of clusters
tabasco(dune, plotree, sp.ind = taxontree)
## No reordering of taxonomy
tabasco(dune, plotree, sp.ind = taxontree, Colv = FALSE)
## Species cluster: most dissimilarity indices do a bad job when
## comparing rare and common species, but Raup-Crick makes sense
sptree <- hclust(vegdist(t(dune), "raup"), "average")
tabasco(dune, plotree, sptree)
```

---

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, stdev = FALSE)
eigengrad(x, w)
## S3 method for class 'wascores'
scores(x, display = c("wa", "stdev", "var", "se", "n2", "raw"), ...)
```

Arguments

x	Environmental variables or ordination scores, or for wascores object with stdev = TRUE.
w	Weights: species abundances.
expand	Expand weighted averages so that they have the same weighted variance as the corresponding environmental variables.
stdev	Estimate weighted standard deviation of WA scores.
display	Type of scores returned.
...	Other arguments passed to functions (currently ignored).

Details

Weighted Averages are a classical way of estimating the species optima along continuous environmental variables (a.k.a. gradients). Function wascores is a simple function that is mainly designed to add species scores to unimodal ordinations (metaMDS, sppscores) or ordering rows or columns to give diagonal pattern of tabulation (vegemite, tabasco). It can also be used to find species "optima" or sampling unit calibrations for community data. For this purpose, specialized packages such **analogue** are recommended (but see calibrate.cca).

First argument of wascores is the variable or a matrix of variables for which weighted averages are needed, and the second argument is the matrix of weights. In classical approaches weights are a community matrix, where taxon abundances define the weights. The number of rows must match. If the first argument is for taxa (columns), community weight matrix must be transposed.

Weighted averages "shrink": they cannot be more extreme than values used for calculating the averages. With expand = TRUE, the function "deshrinks" the weighted averages making their weighted variance equal to the weighted variance of the corresponding input variable. Specialized packages (such as **analogue**) offer a wider range of deshrinking alternatives, but deshrinking can also made after the analysis (see Examples). Function eigengrad returns the strength of expansion as attribute shrinkage of the wascores result for each environmental gradient. The shrinkage equal to the constrained eigenvalue of cca when only this one gradient was used as a constraint, and describes the strength of the gradient.

With stdev = TRUE the function estimates the unbiased weighted standard deviation of the WA estimates using cov.wt. For unbiased standard deviation the virtual number of observations is equal to inverse Simpson index of diversity also known as Hill number N2 (see diversity). The numeric results can be accessed with scores function. Function tolerance uses the same algebra for weighted standard deviation, but bases the variance on linear combination scores (constraints) variables instead of the weighted averages of the sites like wascores.

Weighted averages are closely linked to correspondence analysis (ca, cca). Repeated use of wascores will converge to the first axis of unconstrained correspondence analysis (ca) which therefore is also known as Reciprocal Averaging (Hill 1973). Constrained correspondence analysis (cca) is equivalent to weighted averages and calibrate.cca will return weighted averages of the constraint with different deshrinking.

Value

If stdev = TRUE, function returns an object of class "wascores" with items

wa	A matrix of weighted averages with. If expand=TRUE, attribute shrinkage has the inverses of squared expansion factors or cca eigenvalues for the variable and attribute centre for the weighted means of the variables.
stdev	a matrix of weighted standard deviations
n2	effective sample sizes which are equal to inverse Simpson diversity or Hill number N2

If stdev = FALSE (default), only the plain matrix wa is returned. Function eigengrad returns only the shrinkage attribute. With stdev = TRUE only a brief summary of the result is printed, and the individvual scores can be accessed with scores function.

Author(s)

Jari Oksanen

References

Hill, M.O. (1973) Reciprocal averaging: An eigenvector method of ordination. *Journal of Ecology* 61, 237–249.

See Also

calibrate.cca, tolerance.cca, sppscores.

Examples

```
data(mite, mite.env)
## add species points to ordination
mod <- monoMDS(vegdist(mite))
plot(mod)
## add species points; sppscores does the same and can also add the
## species scores to mod
points(wascores(scores(mod), mite, expand = TRUE), pch="+", col=2)
## Get taxon optima for WatrCont
head(wascores(mite.env$WatrCont, mite))
## WA calibration: site WA from species WA; NB using transpose for site WA
spwa <- wascores(mite.env$WatrCont, mite, expand = TRUE)
wacalib <- wascores(spwa, t(mite), expand = TRUE)
plot(wacalib ~ WatrCont, data=mite.env)
abline(0, 1)
## use traditional 'inverse' regression deshrinking instead of wascores
## 'expand'
wareg <- fitted(lm(WatrCont ~ wacalib, data=mite.env))
head(cbind("WatrCont" = mite.env$WatrCont, "expand" = drop(wacalib),
    "regression" = wareg))
## Reciprocal Averaging algorithm for Correspondence Analysis
## start with random values
u <- runif(nrow(mite))
## repeat the following steps so long that the shrinkage converges
v <- wascores(u, mite, expand = TRUE)
u <- wascores(v, t(mite), expand = TRUE)
```

```
attr(u, "shrinkage") # current estimate of eigenvalue
## The strengths of two continuous variables in the data set
eigengrad(mite.env[, 1:2], mite)
```

---

wcmdscale *Weighted Classical (Metric) Multidimensional Scaling*

Description

Weighted classical multidimensional scaling, also known as weighted *principal coordinates analysis*.

Usage

```
wcmdscale(d, k, eig = FALSE, add = FALSE, x.ret = FALSE, w)
## S3 method for class 'wcmdscale'
plot(x, choices = c(1, 2), display = "sites", type = "t", ...)
## S3 method for class 'wcmdscale'
scores(x, choices = NA, display = "sites", tidy = FALSE, ...)
```

Arguments

d	a distance structure such as that returned by `dist` or a full symmetric matrix containing the dissimilarities.
k	the dimension of the space which the data are to be represented in; must be in $\{1, 2, \ldots, n-1\}$. If missing, all dimensions with above zero eigenvalue.
eig	indicates whether eigenvalues should be returned.
add	an additive constant c is added to the non-diagonal dissimilarities such that all $n-1$ eigenvalues are non-negative. Alternatives are `"lingoes"` (default, also used with `TRUE`) and `"cailliez"` (which is the only alternative in `cmdscale`). See Legendre & Anderson (1999).
x.ret	indicates whether the doubly centred symmetric distance matrix should be returned.
w	Weights of points.
x	The `wcmdscale` result object when the function was called with options `eig = TRUE` or `x.ret = TRUE` (See Details).
choices	Axes to be returned; `NA` returns all real axes.
display	Kind of scores. Normally only `"sites"` are available, but `"species"` can be supplemented with `sppscores`.
type	Type of graph which may be `"t"`ext, `"p"`oints or `"n"`one.
tidy	Return scores that are compatible with **ggvegan**: scores are in a `data.frame`, score type is in the variable `score` labelled as `"sites"` and `"species"` (if available), weights in variable `weigth`, and names in variable `label`.
...	Other arguments passed to graphical functions.

Details

Function wcmdscale is based on function cmdscale (package **stats** of base R), but it uses point weights. Points with high weights will have a stronger influence on the result than those with low weights. Setting equal weights w = 1 will give ordinary multidimensional scaling.

With default options, the function returns only a matrix of scores scaled by eigenvalues for all real axes. If the function is called with eig = TRUE or x.ret = TRUE, the function returns an object of class "wcmdscale" with print, plot, scores, eigenvals and stressplot methods, and described in section Value.

The method is Euclidean, and with non-Euclidean dissimilarities some eigenvalues can be negative. If this disturbs you, this can be avoided by adding a constant to non-diagonal dissimilarities making all eigenvalues non-negative. The function implements methods discussed by Legendre & Anderson (1999): The method of Lingoes (add="lingoes") adds the constant c to squared dissimilarities d using $\sqrt{d^2 + 2c}$ and the method of Cailliez (add="cailliez") to dissimilarities using $d + c$. Legendre & Anderson (1999) recommend the method of Lingoes, and base R function cmdscale implements the method of Cailliez.

Value

If eig = FALSE and x.ret = FALSE (default), a matrix with k columns whose rows give the coordinates of points corresponding to positive eigenvalues. Otherwise, an object of class wcmdscale containing the components that are mostly similar as in cmdscale:

points	a matrix with k columns whose rows give the coordinates of the points chosen to represent the dissimilarities.
eig	the $n - 1$ eigenvalues computed during the scaling process if eig is true.
x	the doubly centred and weighted distance matrix if x.ret is true.
ac, add	additive constant and adjustment method used to avoid negative eigenvalues. These are 0 and FALSE if no adjustment was done.
GOF	Goodness of fit statistics for k axes. The first value is based on the sum of absolute values of all eigenvalues, and the second value is based on the sum of positive eigenvalues
weights	Weights.
negaxes	A matrix of scores for axes with negative eigenvalues scaled by the absolute eigenvalues similarly as points. This is NULL if there are no negative eigenvalues or k was specified, and would not include negative eigenvalues.
call	Function call.

References

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

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

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979). Chapter 14 of *Multivariate Analysis*, London: Academic Press.

See Also

The function is modelled after cmdscale, but adds weights (hence name) and handles negative eigenvalues differently. eigenvals.wcmdscale and stressplot.wcmdscale are some specific methods. Species scores can be added to the result with sppscores.

Examples

```
## Correspondence analysis as a weighted principal coordinates
## analysis of Euclidean distances of Chi-square transformed data
data(dune)
rs <- rowSums(dune)/sum(dune)
d <- dist(decostand(dune, "chi"))
ord <- wcmdscale(d, w = rs, eig = TRUE)
## Ordinary CA
ca <- cca(dune)

## IGNORE_RDIFF_BEGIN
## Eigevalues are numerically similar
ca$CA$eig - ord$eig
## Configurations are similar when site scores are scaled by
## eigenvalues in CA
procrustes(ord, ca, choices=1:19, scaling = "sites")
## IGNORE_RDIFF_END

plot(procrustes(ord, ca, choices=1:2, scaling="sites"))
## Reconstruction of non-Euclidean distances with negative eigenvalues
d <- vegdist(dune)
ord <- wcmdscale(d, eig = TRUE)
## Only positive eigenvalues:
cor(d, dist(ord$points))
## Correction with negative eigenvalues:
cor(d, sqrt(dist(ord$points)^2 - dist(ord$negaxes)^2))
```

Index

∗ **IO**
read.cep, 227
∗ **aplot**
envfit, 92
linestack, 115
ordiarrows, 164
ordihull, 168
ordilabel, 172
ordiplot, 173
ordipointlabel, 176
ordisurf, 181
orditorp, 186
plot.cca, 202
∗ **character**
make.cepnames, 117
∗ **cluster**
cascadeKM, 40
∗ **datagen**
commsim, 56
nullmodel, 154
oecosimu, 158
permat, 190
simulate.rda, 240
∗ **datasets**
BCI, 21
dune, 88
dune.taxon, 89
mite, 133
pyrifos, 217
sipoo, 242
varespec, 269
∗ **distribution**
fisherfit, 99
permustats, 194
radfit, 218
∗ **file**
read.cep, 227
∗ **hplot**
betadisper, 25
biplot.rda, 38
linestack, 115
ordiplot, 173
ordipointlabel, 176
orditorp, 186
plot.cca, 202
vegemite, 282
∗ **htest**
anosim, 14
anova.cca, 16
clamtest, 54
envfit, 92
mantel, 118
mrpp, 141
procrustes, 213
vegan-package, 4
∗ **iplot**
ordiplot, 173
∗ **manip**
beals, 23
decostand, 72
dispweight, 82
optspace, 163
vegemite, 282
∗ **methods**
betadisper, 25
permutest.betadisper, 200
∗ **misc**
vegan-deprecated, 275
∗ **models**
add1.cca, 5
cca.object, 47
deviance.cca, 79
influence.cca, 107
MOStest, 138
nobs.cca, 154
ordistep, 178
simulate.rda, 240
specaccum, 245

SSarrhenius, 255
vegan-package, 4
∗ **multivariate**
add1.cca, 5
adipart, 7
adonis, 11
anosim, 14
anova.cca, 16
avgdist, 19
betadisper, 25
betadiver, 31
bgdispersal, 33
bioenv, 35
cca, 43
cca.object, 47
CCorA, 51
commsim, 56
contribdiv, 62
dbrda, 64
decorana, 68
decostand, 72
designdist, 75
deviance.cca, 79
dispindmorisita, 81
dispweight, 82
distconnected, 84
eigenvals, 90
envfit, 92
eventstar, 96
goodness.cca, 102
goodness.metaMDS, 104
indpower, 106
influence.cca, 107
isomap, 110
kendall.global, 113
mantel, 118
mantel.correlog, 121
MDSrotate, 125
metaMDS, 127
monoMDS, 135
mrpp, 141
mso, 145
multipart, 148
nullmodel, 154
oecosimu, 158
optspace, 163
ordistep, 178
ordisurf, 181
pcnm, 188
permat, 190
permutations, 197
permutest.betadisper, 200
prc, 207
predict.cca, 209
procrustes, 213
rankindex, 221
raupcrick, 225
reorder.hclust, 231
RsquareAdj, 233
scores, 234
screeplot.cca, 235
simper, 238
simulate.rda, 240
spantree, 243
sppscores, 253
stepacross, 257
stressplot.wcmdscale, 259
tsallis, 267
varpart, 270
vegan-package, 4
vegdist, 276
wascores, 285
wcmdscale, 288
∗ **nonparametric**
adonis, 11
anosim, 14
bgdispersal, 33
kendall.global, 113
mrpp, 141
oecosimu, 158
vegan-package, 4
∗ **optimize**
eventstar, 96
∗ **ordination**
CCorA, 51
∗ **package**
vegan-package, 4
∗ **print**
vegemite, 282
∗ **regression**
MOStest, 138
vegan-package, 4
∗ **smooth**
beals, 23
permustats, 194
∗ **spatial**

dispindmorisita, 81
mso, 145
pcnm, 188
vegan-package, 4
∗ **univar**
diversity, 86
fisherfit, 99
nestedtemp, 150
radfit, 218
rarefy, 223
renyi, 229
RsquareAdj, 233
specaccum, 245
specpool, 250
taxondive, 261
treedive, 265
vegan-package, 4
wascores, 285
∗ **utilities**
eventstar, 96
ordiArrowTextXY, 166
.Random.seed, *155*

abbreviate, *117, 118*
add1, *6*
add1.cca, 5, *18*, *46*, *50*, *80*, *179, 180*
add1.default, *6*
adipart, 7, *63, 64*, *87, 88*, *150*, *193*
adonis, 11
adonis2, *15*, *18*, *32, 33*, *50*, *144*, *196*
adonis2 *(adonis)*, 11
agnes, *169, 170*, *244, 245*, *282, 283*
AIC, *79*, *220*
AIC.fitspecaccum *(specaccum)*, 245
AIC.radfit *(radfit)*, 218
alias.cca, *49, 50*
alias.cca *(goodness.cca)*, 102
alias.lm, *103*
anosim, *12, 13*, 14, *144*, *196*, *222, 223*
anova, *18*
anova.betadisper *(betadisper)*, 25
anova.cca, *6*, *11, 12*, 16, *44*, *46*, *50*, *67*, *80*, *179*, *196*, *209*, *274*
anova.glm, *139*
anova.lm, *29*
anova.mlm, *17, 18*
approx, *247*, *284*
ar, *193*
arima, *193*
arrows, *165*, *170*, *174*, *214, 215*
as.factor, *26*
as.fisher *(fisherfit)*, 99
as.hclust, *265*
as.hclust.spantree, *232*
as.hclust.spantree *(spantree)*, 243
as.matrix, *20*
as.mlm.cca, *46*, *50*
as.preston *(fisherfit)*, 99
as.rad *(radfit)*, 218
as.ts.oecosimu *(oecosimu)*, 158
as.ts.permat *(permat)*, 190
avgdist, 19

BCI, 21
beals, 23, *253*
betadisper, *12*, 25, *32, 33*, *154*, *200, 201*
betadistances *(betadisper)*, 25
betadiver, *26*, *28, 29*, 31, *77, 78*, *88*, *152*, *272*, *281*
bgdispersal, 33
bioenv, 35
bioenvdist *(bioenv)*, 35
biplot, *52*
biplot.cca, *50*
biplot.cca *(biplot.rda)*, 38
biplot.CCorA *(CCorA)*, 51
biplot.default, *52*
biplot.rda, 38, *39*, *46*
Box.test, *193*
boxplot, *15*, *28, 29*, *196*, *249*
boxplot.betadisper *(betadisper)*, 25
boxplot.default, *195*
boxplot.permustats *(permustats)*, 194
boxplot.specaccum *(specaccum)*, 245
browseVignettes, *152*
bstick, *70*
bstick *(screeplot.cca)*, 235
bstick.cca, *50*

c.permustats *(permustats)*, 194
ca, *50*, *286*
ca *(cca)*, 43
calibrate *(predict.cca)*, 209
calibrate.cca, *46*, *49, 50*, *286, 287*
calibrate.ordisurf *(ordisurf)*, 181
call, *273*
cancor, *205*

capscale, 6, 16, 17, 33, 46–50, 91, 102–104, 107, 174, 178, 180, 202, 204, 210–212, 240, 241, 253, 254, 259, 260
capscale (dbrda), 64
cascadeKM, 40, 115
cca, 6, 11, 16–18, 43, 47–50, 65–67, 70, 71, 79, 80, 91, 94, 95, 102, 104, 107–109, 145, 154, 166, 169, 170, 174, 178–180, 182, 184, 188, 189, 202–204, 210–212, 233, 237, 238, 240, 241, 259, 260, 271, 272, 279, 286, 287
cca.object, 45, 46, 47, 67, 91, 208
CCorA, 51, 154
chaodist, 278
chaodist (designdist), 75
chisq.test, 59, 160
chull, 170, 171
cIndexKM (cascadeKM), 40
clamtest, 54
cmdscale, 65, 67, 111, 112, 130, 131, 137, 288–290
coef, 49, 220
coef.cca, 46, 49, 50
coef.cca (predict.cca), 209
coef.radfit (radfit), 218
coef.rda, 49
coef.rda (predict.cca), 209
commsim, 8, 56, 83, 84, 149, 155, 157, 159–162, 193
commsimulator, 161
confint.glm, 140
confint.MOStest (MOStest), 138
contour, 181, 184
contr.treatment, 208
contrasts, 45, 208
contribdiv, 62
cooks.distance, 46, 107–109
cooks.distance.cca, 50
cooks.distance.cca (influence.cca), 107
cophenetic, 266
cophenetic.spantree, 259
cophenetic.spantree (spantree), 243
cor, 35, 37, 115, 119–121, 222, 223
cor.test, 120, 222
corresp, 45
cov.wt, 286
coverscale (vegemite), 282
cut, 284
cutree, 115, 231, 232
cutreeord (reorder.hclust), 231

daisy, 36, 222, 266, 280
data.frame, 94, 288
dbrda, 6, 11, 12, 16–18, 33, 47–50, 64, 79, 91, 94, 102–104, 107, 166, 174, 178, 180, 188, 202, 204, 210–212, 253, 254, 259, 260, 272, 274
decobackstand (decostand), 72
decorana, 45, 68, 91, 94, 95, 154, 169, 170, 174, 182, 184, 203, 210, 212, 237, 238, 264, 279
decostand, 25, 45, 72, 109, 133, 220, 224, 271, 274, 276, 277, 279, 280
dendrogram, 232, 282, 283
density, 101, 161, 196
density.default, 195
density.permustats (permustats), 194
Deprecated, 276
designdist, 32, 33, 75, 142, 152, 227, 239, 266, 272, 276, 277, 281
designdist2, 124
designdist2 (designdist), 75
deviance, 79, 220
deviance.cca, 6, 46, 50, 79, 179
deviance.fitspecaccum (specaccum), 245
deviance.radfit (radfit), 218
deviance.rda, 6
deviance.rda (deviance.cca), 79
df.residual, 109
df.residual.cca, 49, 50
df.residual.cca (influence.cca), 107
dispindmorisita, 81
dispweight, 82, 224
dist, 11, 14, 15, 26, 36, 37, 65–67, 78, 85, 86, 89, 127, 142, 146, 226, 243, 245, 257, 258, 261, 262, 266, 270, 272, 280
dist2xy (MDSaddpoints), 124
distconnected, 84, 112, 244, 258, 259
diversity, 8, 64, 86, 101, 230, 231, 253, 263, 268, 286
downweight (decorana), 68
drarefy (rarefy), 223
drop.scope, 17
drop1, 6

drop1.cca, *18*, *46*, *50*, *80*, *179*, *180*
drop1.cca (add1.cca), 5
drop1.default, *6*
dune, 88, *89*
dune.phylodis (dune.taxon), 89
dune.taxon, 89

eigen, *91*
eigengrad (wascores), 285
eigenvals, *49*, *70*, 90, *289*
eigenvals.betadisper (betadisper), 25
eigenvals.cca, *46*, *49*, *50*
eigenvals.decorana, *70*
eigenvals.wcmdscale, *290*
ellipsoidhull, *170*, *171*
envfit, *45*, 92, *103*, *166*, *184*, *185*, *205*, *254*
estaccumR (specpool), 250
estimateR (specpool), 250
estVar, *109*
estVar.cca, *50*
estVar.cca (influence.cca), 107
eventstar, 96
extractAIC, *80*
extractAIC.cca, *6*, *50*, *179*, *180*
extractAIC.cca (deviance.cca), 79

factor, *45*, *94*, *205*, *266*
factorfit (envfit), 92
family, *83*, *139*, *140*, *218–220*
fieller.MOStest (MOStest), 138
fisher.alpha, *100*, *101*, *278*
fisher.alpha (diversity), 86
fisherfit, *87*, 99, *221*, *278*
fitdistr, *100*, *101*
fitspecaccum, *255*, *256*
fitspecaccum (specaccum), 245
fitted, *220*
fitted.capscale (predict.cca), 209
fitted.cca, *48*, *50*, *241*
fitted.cca (predict.cca), 209
fitted.dbrda (predict.cca), 209
fitted.procrustes (procrustes), 213
fitted.radfit (radfit), 218
fitted.rda, *241*
fitted.rda (predict.cca), 209
formula, *11*, *36*, *45*, *49*, *65*, *93*, *94*
friedman.test, *115*

gam, *182–185*
Gamma, *219*, *220*
gaussian, *219*, *233*
gdispweight (dispweight), 82
glm, *83*, *138–140*, *218–220*, *233*
goodness (goodness.cca), 102
goodness.cca, *46*, *50*, 102, *212*
goodness.metaMDS, 104
goodness.monoMDS (goodness.metaMDS), 104

hatvalues, *107–109*
hatvalues.cca, *50*
hatvalues.cca (influence.cca), 107
hatvalues.rda (influence.cca), 107
hclust, *115*, *142*, *143*, *169*, *170*, *231*, *232*, *244*, *245*, *265*, *266*, *282–284*
heatmap, *282–284*
hiersimu, *150*, *193*
hiersimu (adipart), 7
how, *6*, *11*, *14–18*, *51*, *53*, *93*, *95*, *119*, *120*, *142*, *144*, *146*, *169*, *179*, *197*, *199–201*, *214*, *216*, *229*, *238*, *246*, *250*, *267*, *271*

identify.ordiplot, *28*, *39*, *172*, *205*, *215*, *220*
identify.ordiplot (ordiplot), 173
image, *181*, *282*, *284*
indpower, *25*, 106
inertcomp, *46*, *50*, *274*
inertcomp (goodness.cca), 102
influence.cca, *50*, 107
influence.measures, *109*
initMDS (metaMDS), 127
intersetcor, *46*, *50*
intersetcor (goodness.cca), 102
invisible, *169*, *170*, *225*
isomap, 110, *154*, *259*
isomapdist (isomap), 110
isoMDS, *105*, *129*, *131*, *136–138*, *235*

kendall.global, 113, *197*
kendall.post, *197*
kendall.post (kendall.global), 113
kmeans, *40–42*, *115*

labels.cca, *50*
labels.cca (plot.cca), 202
labels.envfit (envfit), 92
labels.ordipointlabel (ordipointlabel), 176

lag.plot, *193*
Lattice, *220, 221*
lda, *125, 126*, *205*
legend, *54*, *146*, *208*
lines, *165*, *169–171*, *224*
lines.fitspecaccum *(*specaccum*)*, 245
lines.permat *(*permat*)*, 190
lines.preston *(*fisherfit*)*, 99
lines.prestonfit *(*fisherfit*)*, 99
lines.procrustes *(*procrustes*)*, 213
lines.radfit *(*radfit*)*, 218
lines.radline *(*radfit*)*, 218
lines.renyiaccum *(*renyi*)*, 229
lines.spantree *(*spantree*)*, 243
lines.specaccum *(*specaccum*)*, 245
linestack, 115, *208*
lm, *46*, *107*, *109*, *188*, *233*, *241*
lm.influence, *46*
locator, *273*
logLik, *220*
logLik.fitspecaccum *(*specaccum*)*, 245
logLik.radfit *(*radfit*)*, 218
lset, *221*

Machine, *223*
make.cepnames, *89*, 117
make.commsim, *8*, *149*, *157*, *159, 160*, *191–193*
make.commsim *(*commsim*)*, 56
make.names, *228*
mantel, *13*, *15*, *32, 33*, *37*, 118, *121, 122*, *144*, *146*, *196*, *216*, *222, 223*
mantel.correlog, *120*, 121, *146, 147*
mantel.partial, *37*, *122*, *189*, *196*
matlines, *230*, *249*
matplot, *208*, *230*
mcnemar.test, *34*
MDSaddpoints, 124
MDSrotate, 125, *130*, *132, 133*, *137*
meandist, *240*
meandist *(*mrpp*)*, 141
metaMDS, *33*, *37*, *45*, *65, 66*, *71*, *105*, *112*, *124–126*, 127, *137, 138*, *154*, *253, 254*, *259*, *286*
metaMDSdist, *65, 66*
metaMDSdist *(*metaMDS*)*, 127
metaMDSiter *(*metaMDS*)*, 127
metaMDSredist *(*metaMDS*)*, 127
mite, 133
model.frame.cca, *50*
model.frame.cca *(*cca.object*)*, 47
model.matrix, *44*
model.matrix.cca, *50*
model.matrix.cca *(*cca.object*)*, 47
model.matrix.rda *(*cca.object*)*, 47
monoMDS, *14*, *37*, *71*, *105*, *124–133*, 135, *216*, *223*, *253, 254*, *259*
MOStest, 138
mrpp, *12, 13*, *15*, 141, *196*
mso, *50*, *122*, 145
msoplot *(*mso*)*, 145
multipart, *10*, *87, 88*, 148
mvrnorm, *241*

na.action, *49*
na.exclude, *44*, *66*
na.fail, *12*, *44*, *66*
na.omit, *44*, *66*
names<-.envfit *(*envfit*)*, 92
nestedbetajac, *33*
nestedbetajac *(*nestedtemp*)*, 150
nestedbetasor, *33*
nestedbetasor *(*nestedtemp*)*, 150
nestedchecker, *58*, *160*
nestedchecker *(*nestedtemp*)*, 150
nesteddisc, *160*
nesteddisc *(*nestedtemp*)*, 150
nestedn0, *160*
nestedn0 *(*nestedtemp*)*, 150
nestednodf, *159, 160*
nestednodf *(*nestedtemp*)*, 150
nestedtemp, 150, *159–162*
nlm, *101*, *219*, *224*
nls, *247–249*, *255, 256*
no.shared, *128*, *222, 223*, *258, 259*
no.shared *(*distconnected*)*, 84
nobs.betadisper *(*nobs.cca*)*, 154
nobs.cca, *49, 50*, 154
nobs.CCorA *(*nobs.cca*)*, 154
nobs.decorana *(*nobs.cca*)*, 154
nobs.fitspecaccum *(*specaccum*)*, 245
nobs.isomap *(*nobs.cca*)*, 154
nobs.metaMDS *(*nobs.cca*)*, 154
nobs.pcnm *(*nobs.cca*)*, 154
nobs.procrustes *(*nobs.cca*)*, 154
nobs.rad *(*nobs.cca*)*, 154
nobs.varpart *(*nobs.cca*)*, 154
nobs.wcmdscale *(*nobs.cca*)*, 154
nullmodel, *56*, *61*, *83*, *152*, 154, *159–162*, *193*

numPerms, *199, 200*

object.size, *160*
oecosimu, *8–10*, *57*, *61*, *149–153*, 158, *193*, *196*, *226, 227*, *266*
optim, *177*
optimize, *97*
optspace, *74*, 163
ordConstrained, *50*
ordConstrained *(*cca.object*)*, 47
ordered, *266*
orderingKM *(*cascadeKM*)*, 40
ordiareatest, *196*
ordiareatest *(*ordihull*)*, 168
ordiArrowMul, *95*
ordiArrowMul *(*ordiArrowTextXY*)*, 166
ordiarrows, 164, *171*
ordiArrowTextXY, 166
ordibar *(*ordihull*)*, 168
ordicluster, *244*
ordicluster *(*ordihull*)*, 168
ordiellipse *(*ordihull*)*, 168
ordigrid, *171*
ordigrid *(*ordiarrows*)*, 164
ordihull, 168
ordilabel, *94*, *111*, *165*, *169*, 172, *174, 175*, *205*, *224*
ordimedian *(*betadisper*)*, 25
ordiplot, *28*, *32*, *39*, *111*, *133*, *164, 165*, *168*, *172*, 173, *186, 187*, *205, 206*, *220*, *235*
ordipointlabel, *172*, *174, 175*, 176, *203*, *205*
ordiR2step, *6*, *46*, *50*
ordiR2step *(*ordistep*)*, 178
ordisegments, *171*
ordisegments *(*ordiarrows*)*, 164
ordispider, *103*
ordispider *(*ordihull*)*, 168
ordistep, *6*, *18*, *46*, *50*, 178
ordisurf, *45*, *92*, *95*, 181
orditorp, *172*, 186
ordixyplot, *235*
ordiYbar *(*cca.object*)*, 47

p.adjust, *113, 114*, *121*
pairs, *196*
pairs.permustats *(*permustats*)*, 194
par, *224*, *246*
pca, *50*
pca *(*cca*)*, 43
pchisq, *83*
pcnm, *91*, *154*, 188
pco, *50*, *253*
pco *(*dbrda*)*, 64
permat, 190
permatfull, *61*, *157*, *159, 160*
permatfull *(*permat*)*, 190
permatswap, *59–61*, *157*, *159, 160*
permatswap *(*permat*)*, 190
permustats, *15*, *18*, *120*, *144*, *161*, 194, *216*
permutations, *15*, *94*, *119*, 197
permutest, *200*, *233*
permutest *(*anova.cca*)*, 16
permutest.betadisper, *28, 29*, *196*, 200
permutest.cca, *49, 50*, *196*
persp, *181*, *229, 230*, *269*
persp.renyiaccum *(*renyi*)*, 229
persp.tsallisaccum *(*tsallis*)*, 267
phyper, *278*
plot, *116*, *224*
plot.anosim *(*anosim*)*, 14
plot.betadisper *(*betadisper*)*, 25
plot.betadiver *(*betadiver*)*, 31
plot.cascadeKM *(*cascadeKM*)*, 40
plot.cca, *38, 39*, *45, 46*, *50*, *67*, *93*, *164*, *168*, *173–175*, *177*, *187*, 202
plot.clamtest *(*clamtest*)*, 54
plot.contribdiv *(*contribdiv*)*, 62
plot.decorana, *164*, *168*, *187*
plot.decorana *(*decorana*)*, 68
plot.default, *27*, *187*, *237*
plot.envfit *(*envfit*)*, 92
plot.fisher *(*fisherfit*)*, 99
plot.fisherfit *(*fisherfit*)*, 99
plot.fitspecaccum *(*specaccum*)*, 245
plot.gam, *181*, *184*
plot.hclust, *232*
plot.isomap *(*isomap*)*, 110
plot.lm, *139, 140*
plot.mantel.correlog *(*mantel.correlog*)*, 121
plot.meandist *(*mrpp*)*, 141
plot.metaMDS, *187*
plot.metaMDS *(*metaMDS*)*, 127
plot.monoMDS *(*monoMDS*)*, 135
plot.MOStest *(*MOStest*)*, 138
plot.nestednodf *(*nestedtemp*)*, 150

plot.nestedtemp (nestedtemp), 150
plot.ordipointlabel (ordipointlabel), 176
plot.ordisurf (ordisurf), 181
plot.permat (permat), 190
plot.poolaccum (specpool), 250
plot.prc (prc), 207
plot.preston (fisherfit), 99
plot.prestonfit (fisherfit), 99
plot.procrustes, *174*
plot.procrustes (procrustes), 213
plot.rad (radfit), 218
plot.radfit (radfit), 218
plot.radline (radfit), 218
plot.renyi, *269*
plot.renyi (renyi), 229
plot.renyiaccum, *269*
plot.renyiaccum (renyi), 229
plot.spantree (spantree), 243
plot.specaccum (specaccum), 245
plot.taxondive (taxondive), 261
plot.varpart (varpart), 270
plot.varpart234 (varpart), 270
plot.wcmdscale (wcmdscale), 288
plotmath, *116*
points, *165*, *177*, *186*, *187*
points.cca, *50*
points.cca (plot.cca), 202
points.decorana (decorana), 68
points.metaMDS (metaMDS), 127
points.monoMDS (monoMDS), 135
points.ordiplot, *204*
points.ordiplot (ordiplot), 173
points.procrustes (procrustes), 213
points.radfit (radfit), 218
points.radline (radfit), 218
polygon, *169–172*, *249*
poolaccum, *249*
poolaccum (specpool), 250
postMDS (metaMDS), 127
prc, *46*, 207
prcomp, *91*, *235*, *237*, *238*, *259*
predict, *220*
predict.cca, *46*, *48–50*, *124*, 209, *241*
predict.dbrda (predict.cca), 209
predict.decorana, *71*
predict.decorana (predict.cca), 209
predict.fitspecaccum (specaccum), 245
predict.gam, *183*, *184*
predict.nls, *248*
predict.procrustes (procrustes), 213
predict.radfit (radfit), 218
predict.radline (radfit), 218
predict.rda, *49*, *241*
predict.rda (predict.cca), 209
predict.specaccum (specaccum), 245
pregraphKM (cascadeKM), 40
prestondistr (fisherfit), 99
prestonfit, *221*
prestonfit (fisherfit), 99
princomp, *91*, *235*, *237*, *238*, *259*
print.betadisper (betadisper), 25
print.cca, *50*
print.cca (cca.object), 47
print.commsim (commsim), 56
print.default, *27*
print.nullmodel (nullmodel), 154
print.permat (permat), 190
print.simmat (nullmodel), 154
print.specaccum (specaccum), 245
print.summary.cca (plot.cca), 202
print.summary.permat (permat), 190
procrustes, *37*, *130*, *133*, *154*, 213, *235*
profile.glm, *140*
profile.MOStest (MOStest), 138
protest, *37*, *196*, *222*, *223*
protest (procrustes), 213
pyrifos, 217

qqnorm, *161*, *196*, *221*
qqnorm.permustats (permustats), 194
qqplot, *101*, *221*
qr, *49*
qr.cca, *49*, *50*
qr.cca (influence.cca), 107
quasipoisson, *83*, *220*

r2dtable, *59*, *191–193*
rad.lognormal, *100*
rad.lognormal (radfit), 218
rad.null (radfit), 218
rad.preempt (radfit), 218
rad.zipf (radfit), 218
rad.zipfbrot (radfit), 218
radfit, *101*, *154*, 218, *237*
radlattice (radfit), 218
rank, *15*, *222*

rankindex, *37*, *130*, *133*, 221
rarecurve (rarefy), 223
rarefy, *88*, 223, *247*, *249*
rareslope (rarefy), 223
raupcrick, *77, 78*, *160*, *162*, 225, *272*, *278*
rda, *6*, *11, 12*, *16–18*, *38, 39*, *47–50*, *64–67*, *79, 80*, *91*, *94, 95*, *102–104*, *107*, *109*, *145*, *166*, *170*, *174*, *178–180*, *188*, *202*, *204*, *207–212*, *233*, *240, 241*, *259, 260*, *272*, *274*
rda (cca), 43
read.cep, 227
read.fortran, *228*
relevel, *208*
renyi, *88*, *149*, 229, *268, 269*
renyiaccum, *249*, *268, 269*
renyiaccum (renyi), 229
reorder, *231*
reorder.dendrogram, *231, 232*
reorder.hclust, 231
residuals, *220*
residuals.cca, *48*, *50*
residuals.cca (predict.cca), 209
residuals.glm, *220*
residuals.procrustes (procrustes), 213
rev.hclust (reorder.hclust), 231
RNGversion, *199*
rnorm, *241*
rrarefy, *20*, *249*
rrarefy (rarefy), 223
RsquareAdj, *179, 180*, 233, *272*, *274*
RsquareAdj.cca, *50*
RsquareAdj.rda, *46*
rstandard, *107–109*
rstandard.cca, *50*
rstandard.cca (influence.cca), 107
rstudent, *107–109*
rstudent.cca, *50*
rstudent.cca (influence.cca), 107
rug, *116*

s, *183, 184*
sammon, *244*
sample, *193*, *241*
scale, *36*
scores, *29*, *49*, *92*, *94*, *111*, *165–167*, *171, 172*, *174*, *181–183*, *185*, *187*, *214, 215*, 234, *243*, *283*
scores.betadisper (betadisper), 25
scores.betadiver (betadiver), 31
scores.cca, *46*, *49, 50*, *94, 95*, *103*, *211*, *214, 215*, *234, 235*, *264*
scores.cca (plot.cca), 202
scores.decorana, *234, 235*
scores.decorana (decorana), 68
scores.envfit, *235*
scores.envfit (envfit), 92
scores.hclust (reorder.hclust), 231
scores.metaMDS, *234, 235*
scores.metaMDS (metaMDS), 127
scores.monoMDS, *235*
scores.monoMDS (monoMDS), 135
scores.ordihull (ordihull), 168
scores.ordiplot (ordiplot), 173
scores.pcnm, *235*
scores.pcnm (pcnm), 188
scores.rda, *38*, *49*, *94*, *207*, *211*, *235*
scores.rda (plot.cca), 202
scores.wascores (wascores), 285
scores.wcmdscale (wcmdscale), 288
screeplot, *238*
screeplot.cca, *46*, *50*, 235
screeplot.decorana (screeplot.cca), 235
screeplot.prcomp (screeplot.cca), 235
screeplot.princomp (screeplot.cca), 235
segments, *165*, *169–171*, *214, 215*, *249*
selfStart, *248*
set.seed, *155*
Shepard, *104, 105*
showvarparts (varpart), 270
shuffleSet, *199, 200*, *238*, *241*
sigma, *109*
sigma.cca, *50*
sigma.cca (influence.cca), 107
simmat (nullmodel), 154
simper, 238
simpleDBRDA (varpart), 270
simpleRDA2 (varpart), 270
simpson.unb (diversity), 86
simulate, *154*, *240, 241*
simulate.capscale (simulate.rda), 240
simulate.cca, *46*, *50*
simulate.cca (simulate.rda), 240
simulate.nullmodel, *61*, *159, 160*, *162*, *191*, *241*
simulate.nullmodel (nullmodel), 154
simulate.rda, 240

sipoo, 242
smbind (nullmodel), 154
smooth.terms, *182, 183*
spandepth (spantree), 243
spantree, *86*, *112*, *189*, 243
specaccum, *225*, *230, 231*, 245, *252, 253*
specnumber (diversity), 86
specpool, *25*, *100, 101*, *225*, *246, 247*, 250, *278*
specpool2vect (specpool), 250
specslope (specaccum), 245
spenvcor, *46*
spenvcor (goodness.cca), 102
spline, *247*
sppscores, *49*, *65–67*, *131*, *133*, *136, 137*, 253, *286–288*, *290*
sppscores<- (sppscores), 253
sqrt, *19*, *129*
SSarrhenius, *248*, 255
SSasymp, *248*
SSD, *109*
SSD.cca, *50*
SSD.cca (influence.cca), 107
SSgitay, *248*
SSgitay (SSarrhenius), 255
SSgleason, *248*
SSgleason (SSarrhenius), 255
SSgompertz, *248*, *255*
SSlogis, *248*, *255*
SSlomolino, *248*
SSlomolino (SSarrhenius), 255
SSmicmen, *248*, *255*
SSweibull, *248*, *255*
step, *6*, *46*, *79, 80*, *178–180*
stepacross, *24*, *65, 66*, *84–86*, *111, 112*, *128*, *130*, *132, 133*, *222, 223*, *244*, 257
str.nullmodel (nullmodel), 154
stressplot, *132, 133*, *137*, *259, 260*, *289*
stressplot (goodness.metaMDS), 104
stressplot.capscale (stressplot.wcmdscale), 259
stressplot.cca, *50*, *105*
stressplot.cca (stressplot.wcmdscale), 259
stressplot.dbrda (stressplot.wcmdscale), 259
stressplot.monoMDS, *260*
stressplot.prcomp (stressplot.wcmdscale), 259
stressplot.princomp (stressplot.wcmdscale), 259
stressplot.rda (stressplot.wcmdscale), 259
stressplot.wcmdscale, *105*, 259, *290*
strheight, *167*
stripchart, *116*
strwidth, *167*
summary.anosim (anosim), 14
summary.bioenv (bioenv), 35
summary.cca, *45, 46*, *50*
summary.cca (plot.cca), 202
summary.clamtest (clamtest), 54
summary.decorana (decorana), 68
summary.dispweight (dispweight), 82
summary.eigenvals (eigenvals), 90
summary.glm, *139*
summary.isomap (isomap), 110
summary.mantel (mantel), 118
summary.meandist (mrpp), 141
summary.mrpp (mrpp), 141
summary.oecosimu (oecosimu), 158
summary.ordiareatest (ordihull), 168
summary.ordiellipse (ordihull), 168
summary.ordihull (ordihull), 168
summary.permat (permat), 190
summary.permustats, *119*, *143*, *160*, *170*
summary.permustats (permustats), 194
summary.poolaccum (specpool), 250
summary.prc (prc), 207
summary.procrustes (procrustes), 213
summary.radfit.frame (radfit), 218
summary.simper (simper), 238
summary.specaccum (specaccum), 245
summary.taxondive (taxondive), 261
summary.varpart (varpart), 270
svd, *44*, *91*
swan, *259*
swan (beals), 23
symbols, *95*

tabasco, *286*
tabasco (vegemite), 282
taxa2dist (taxondive), 261
taxondive, *90*, 261, *267*
te, *184*
terms, *49*
text, *165*, *167*, *169*, *172*, *174*, *177*, *186, 187*

text.cca, *50*
text.cca (plot.cca), 202
text.decorana (decorana), 68
text.metaMDS (metaMDS), 127
text.monoMDS (monoMDS), 135
text.ordiplot, *173*, *177*, *204*
text.ordiplot (ordiplot), 173
text.procrustes (procrustes), 213
toCoda (oecosimu), 158
toCoda.permat (permat), 190
tolerance, *70*, 263, *286*
tolerance.cca, *46*, *50*, *287*
treedist, *90*
treedist (treedive), 265
treedive, *90*, *160*, *162*, 265
treeheight (treedive), 265
try, *157*
tryCatch, *157*
ts, *161*
tsallis, *88*, *97*, *98*, *149*, 267
tsallisaccum, *249*
tsallisaccum (tsallis), 267
tsdiag, *193*
TukeyHSD, *27–29*
TukeyHSD.betadisper, *201*
TukeyHSD.betadisper (betadisper), 25

update, *154*
update.nullmodel (nullmodel), 154

varechem (varespec), 269
varespec, 269
varpart, *13*, *46*, *50*, *154*, *234*, 270
varpart2 (varpart), 270
varpart3 (varpart), 270
varpart4 (varpart), 270
vcov, *107*, *109*
vcov.cca, *50*
vcov.cca (influence.cca), 107
vectorfit (envfit), 92
vegan (vegan-package), 4
vegan-deprecated, 275
vegan-package, 4
vegdist, *11*, *12*, *14*, *15*, *19*, *20*, *26*, *32*, *33*, *36*, *37*, *65–67*, *72*, *73*, *76–78*, *85*, *86*, *127*, *130*, *131*, *133*, *142–144*, *222*, *223*, *226*, *227*, *239*, *243*, *245*, *257*, *258*, *266*, *272*, 276
vegemite, 282, *286*
veiledspec, *253*
veiledspec (fisherfit), 99
vif.cca, *46*, *49*, *50*
vif.cca (goodness.cca), 102
vignette, *47*, *152*, *204*, *252*

wascores, *128*, *131–133*, *136*, *212*, *254*, *282–284*, 285
wcmdscale, *11*, *26*, *65*, *67*, *73*, *91*, *137*, *154*, *189*, *253*, *254*, *259*, *260*, *271*, *279*, 288
weighted.mean, *232*
weights.cca, *50*
weights.cca (cca.object), 47
weights.decorana, *70*
weights.decorana (cca.object), 47
weights.rda (cca.object), 47
wisconsin, *129*, *133*
wisconsin (decostand), 72

xy.coords, *237*
xyplot, *221*