BIOSTATS Documentation

BIOSTATS is a collection of R functions written to aid in the statistical analysis of ecological data sets using both univariate and multivariate procedures. All of these functions make use of existing R functions and many are simply convenience wrappers for functions contained in other popular libraries, such as VEGAN and LABDSV for multivariate statistics, however others are custom functions written using functions in the base or stats libraries.

Author:

Kevin McGarigal, Professor
Department of Environmental Conservation
University of Massachusetts, Amherst

Date Last Updated:

9 February 2018

Functions:

all.subsets.gam. . . . 3
all.subsets.glm. . . . 6
box.plots. . . . 13
by.names. . . . 15
ci.lines. . . . 17
class.monte. . . . 18
clus.composite. . . . 20
clus.stats. . . . 22
cohen.kappa. . . . 24
col.summary. . . . 26
commission.mc. . . . 28
concordance. . . . 30
contrast.matrix. . . . 34
cov.test. . . . 36
data.dist. . . . 38
data.stand. . . . 42
data.trans. . . . 45
dist.plots. . . . 49
distributions. . . . 51
drop.var. . . . 54
edf.plots. . . . 56
ecdf.plots. . . . 58
foa.plots. . . . 60
hclus.cophenetic. . . . 63
hclus.scree. . . . 65
hclus.table. . . . 67
hist.plots. . . . 69
intrasetcor. . . . 71

kappa.mc. 73
lda.structure. 75
mantel2. 77
mantel.part. 79
mrpp2. 83
mv.outliers. 85
nhclus.scree. 88
nmds.monte. 90
nmds.scree. 92
norm.test. 94
ordi.monte. 96
ordi.overlay. 98
ordi.part. 100
ordi.scree. 106
pca.communality. 108
pca.eigenval. 110
pca.eigenvec. 112
pca.structure. 114
plot.anosim. 116
plot.mantel. 118
plot.mrpp. 120
plot.ordi.part. 122
qqnorm.plots. 124
ran.split. 126
redun.plot. 128
replace.missing. 130
scatter.plots. 131
sum.stats. 133
tau. 136
uv.outliers. 138
uv.plots. 140

Function: all.subsets.gam

Description:

Fits a generalized additive model (using the gam function in the mgcv library) to all possible subsets of the predictor variables, computes a variety of goodness-of-fit statistics including adjusted R2, deviance explained, AICc, deltaAICc, and AICc model weights for each model, and calculates a measure of variable importance for the set of predictors.

Usage:

all.subsets.gam(y, x.smooth, x.parametric=NULL, family=binomial(link=logit), maxp=5, select='all', delta=7, rank=10, ...)

Dependency:

library(mgcv); library(MuMIn)

Arguments:

y: *required* vector containing the dependent (response) variable.

x.smooth: *required* data frame containing one or more continuous, independent (predictor) variables to be fit with smooth functions. Also, if a single variable is selected from a data frame using a column index, then you use the drop=FALSE argument to carry the column name into the output tables (e.g., data[,3,drop=FALSE]). Alternatively, use the subset() function to select the smooth predictor variables.

x.parametric: *optional* data frame containing one or more continuous or categorical, independent (predictor) variables to be fit with parametric functions. Also, if a single variable is selected from a data frame using a column index, then you use the drop=FALSE argument to carry the column name into the output tables (e.g., data[,3,drop=FALSE]). Alternatively, use the subset() function to select the parametric predictor variables.

family: *optional* error family and link function for the generalized linear model (e.g., gaussian, binomial, poisson). [default = binomial(link=logit)]

maxp: *optional* maximum number of terms (excluding the intercept) to include in the model subsets. Note, this can be less than the number of independent variables considered. [default = 5]

select: *optional* limit on the number of models included in the summary table. Choices are:
'all' = include all models in the table
'delta' = include only models in the table with a delta AICc less the specified delta (see below)
'rank' = include only models in the table with a rank less than the specified rank

(see below)
Note, this argument not only effects the list of models included in the model summary table, but it also effects the variable importance measure based on model AICc weights (see value below), as only the models included in this table will be considered. [default = 'all']

delta: *optional* delta AICc for limiting the models displayed in the model summary table and used in the variable importance measure based on model AICc weights, if select='delta'. [default = 7]

rank: *optional* rank for limiting the models displayed in the model summary table and used in the variable importance measure based on model AICc weights, if select = 'rank'. [default = 10]

... *optional* additional arguments to be passed to the gam() function in the mgcv library.

Details:

all.subsets.gam is simply a convenience wrapper for the gam() function designed to facilitate model comparison and evaluate the relative importance of predictors. Currently, this function is limited to generalized additive models fit using the gam() function in the mgcv library. Model comparison is based on AICc, delta AICc and AICc model weight. Variable importance is calculated as the sum of AICc model weights for each variable (i.e., the sum of model weights across all models containing each variable). An important variable is one that is in all the models with the greatest support based on AICc.

In addition, all smooth predictors (specified with x.smooth=) must be numeric (or integer) variables appropriate for smoothing. If the numeric variable contains too few discrete values (<10), the default gam model will fail. This is because the basis for the smooth function is 10 by default and this requires that more than 10 discrete values exist. The gam() function provides options for controlling the basis for each smooth term, but this is best done individually for each term (see help(gam) for more details on the use of the gam function). Currently, the all.subsets.gam() function does not allow you to modify the basis for each smooth term. Thus, you must be content with the default basis for each term. In addition, including factors in the x.smooth set of predictors will cause the function to fail, as the function will attempt to smooth all the specified terms, and factors cannot be smoothed. Factors must be included as parametric terms using the x.parametric argument.

NOTE, processing time increases dramatically with increasing number of variables and sample size.

Value:

all.subsets.gam returns a list containing the following two components:

modelAIC: Data frame containing a single row for each model and the following columns:
Model: list of the independent variables (predictors)

adjR2: adjusted R^2 computed by gam()
D2: proportion of deviance explained by the model
AICc: corrected AIC
Delta: difference between model AICc and the minimum AICc
Wgt: model AICc weight
Rank: model rank order based on AICc

variable.importance: Data frame containing a row for each independent variable in the data set and the following columns:
Term: independent variable
N: number of retained models (based on select =) containing the term
AICc: variable importance measure based on the sum of AICc model weights across models containing the variable

Author:

K. McGarigal, April 12, 2008;
Feb 28, 2014, substituted AICc computation with the one from MuMIn package based on suggestion by Ivor Williams

References:

See help files for gam().

Examples:

```
x<-read.csv('turtle.csv', header=TRUE)
all.subsets.gam(x[,2], x.smooth=x[,5:10], x.parametric=x[,11:13],family=binomial, maxp=3)
```

Function: all.subsets.glm

Description:

Fits a generalized linear model (using the glm function) to all possible subsets of the predictor variables (optionally forcing one or more variables into all models and including quadratic terms for select variables), computes a variety of goodness-of-fit statistics including deviance explained, AICc, deltaAICc, and AICc model weights for each model, and either ranks models based on AICc or, if the model is a binary logistic regression model (i.e, binary response variable), either the false positive (commission) error rate given a specified sensitivity (proportion of true presences predicted to be present) or Kappa (chance-corrected correct classification rate). In addition, the function optionally calculates one or more measures of variable importance for the set of predictors and optionally computes modeling averaging based on the model.avg() function in the MuMIn package.

Usage:

```
all.subsets.glm(y, x1=NULL,x2=NULL,x2.linear=FALSE,x.force=NULL,
family=binomial(link=logit), offset=NULL, weights=NULL, maxp=5, select='AIC',
sensitivity=.95, delta.AIC=7, delta.FP=.1, delta.Kappa=.1,rank=10, varimp=FALSE,
gof='logLik', coef.table=FALSE,model.ave=FALSE,...)
```

Dependency:

library(hier.part) if varimp = TRUE
library(MuMIn) if model.ave=TRUE

Arguments:

y: *required* vector containing the dependent (response) variable.

x1: *optional* data frame containing one or more independent (predictor) variables for which only linear terms will be considered. Note, if x1 is null, then x2 must contain two or more predictors. Also, if a single variable is selected from a data frame using a column index, then you use the drop=FALSE argument to carry the column name into the output tables (e.g., data[,3,drop=FALSE]). Alternatively, use the subset() function to select the predictor variables. [default=NULL]

x2: *optional* data frame containing one or more independent (predictor) variables for which quadratic terms will be considered. Note, this does not guarantee that when the quadratic term is in the model that the corresponding linear term will also be in the model, but see the x2.linear argument below. Note, if x2 is null, then x1 must contain two or more predictors. Also, if a single variable is selected from a data frame using a column index, then you use the drop=FALSE argument to carry the column name into the output tables (e.g., data[,3,drop=FALSE]). Alternatively, use the subset() function to select the predictor variables. [default=NULL]

x2.linear: *logical* (TRUE or FALSE) indicating whether to force the inclusion of the linear term when including the corresponding quadratic term in the model for the variables

included in x2. Note, this guarantees that when the quadratic term is in the model that the corresponding linear term will also be in the model. This argument is ignored if x2 is null. [default=FALSE]

x.force: *optional* data frame containing one or more independent (predictor) variables to force enter into every model. Note, if you want to force quadratic terms in the model, with or without their corresponding linear terms, these variables must be derived separately and entered here in the list of forced variables. Note, if a single variable is selected from a data frame using a column index, then use the drop=FALSE argument to carry the column name into the output tables (e.g., data[,3,drop=FALSE]). Alternatively, use the subset() function to select the forcing variables. [default = NULL]

family: *optional* error family and link function for the generalized linear model (e.g., gaussian, binomial, poisson). [default = binomial(link=logit)]

offset: *optional* offset term (vector) for the model, typically used to adjust observations for unequal sampling effort. [default = NULL]

weights: *optional* weights term (vector) for the model, used to adjust the weight of each observation for various reasons. [default = NULL]

maxp: *optional* maximum number of terms (excluding the intercept, any forced predictors, and the linear term corresponding to a quadratic term if x2.linear=TRUE) to include in the model subsets. Note, this can be less than the number of independent variables considered. [default = 5]

select: *optional* choice of whether to rank models based on AICc, false positive (commission) error rate, or Cohen's Kappa statistic for inclusion in the output table. Choices are:

'AIC' = rank models based on AICc (in ascending order).

'commission' = rank models based on false positive (commission) error rate based on the specified sensitivity level (see below)(in ascending order). Note, this method is only suitable for binary logistic regression when the response is presence/absence.

'Kappa' = rank models based on maximizing cohen's Kappa statistic (chance-corrected correct classification rate)(in descending order). Note, this method is only suitable for binary logistic regression when the response is presence/absence.

Note, this argument not only effects the list of models included in the model summary table, but it also effects the variable importance measure based on model AICc weights (see value below), as only the models included in this table will be considered, if varimp = TRUE. [default = 'AIC']

sensitivity: *optional* sensitivity level (proportion of true positives predicted to be positive) for determining the cut point for predicting observations as 0 (absent) or 1 (present); ranges from 0-1; only applicable if select = 'commission'. [default = .95]

delta.AIC: *optional* delta AICc threshold for limiting the models displayed in the model summary table; include only models with less than or equal to the specified deltaAIC up to the limit in the number of models specified by the rank argument; also limits the models considered in the variable importance measure if varimp = TRUE; only applicable if select = 'AIC'. [default = 7]

delta.FP: *optional* delta false positive (FP) rate threshold for limiting the models displayed in the model summary table; include only models with less than or equal to the specified deltaFP; also limits the models considered in the variable importance measure if varimp = TRUE; only applicable if select = 'commission'. [default = .1]

delta.Kappa: *optional* delta Kappa statistic for limiting the models displayed in the model summary table; include only models with less than or equal to the specified deltaFP; also limits the models considered in the variable importance measure if varimp = TRUE; only applicable if select = 'Kappa'. [default = .1]

rank: *optional* rank for limiting the models displayed in the model summary table and is based either on the rank order delta.AICc (if select='AIC'), rank order delta.FP (if select = 'commission') or rank order delta.Kappa (if select='Kappa), but it only applies if the limit set by rank results in fewer models than the limit set by the selected delta metric above (i.e., the limit is set by the minium of the rank and the number determined by the delta threshold); also used to limit the models considered in the variable importance measure based on model AICc weights (see below) if varimp=TRUE. [default = 10]

varimp: *optional* logical (TRUE or FALSE) for determining whether to compute variable importance measures. [default = FALSE]

gof: *optional* goodness-of-fit statistic to use in the evaluation of variable importance using the hier.part() function. Choices are:
'RMSPE' = Root-mean-square 'prediction' error
'logLik' = Log-Likelihood
'Rsqu' = R-squared.
Note, this argument should be selected based on the error family selected. [default = 'logLik']

coef.table: *optional* logical (TRUE or FALSE) indicating whether to include the complete list of parameter coefficients and their associated statistics (standard error, Z, and p-value) for all of the models included in the final summary (based on the select, deltaAIC, deltaFP and rank arguments. [default = FALSE]

model.ave: *optional* logical (TRUE or FALSE) indicating whether to conduct model averaging using the MuMIn package. If TRUE, both the subset and full model-averaged coefficients and their associated standard errors, z-values and corresponding *p*-values are computed [default = FALSE]

... *optional* additional arguments to be passed to the glm() or hier.part() functions.

Details:

all.subsets.glm() is simply a convenience wrapper for a variety of functions designed to facilitate model comparison and evaluate the relative importance of predictors. Currently, this function is limited to generalized linear models fit using the glm() function. Model comparison is based on one of three approaches: 1) model AICc, delta AICc and AICc model weights; 2) model false positive (commission error) rate; or 3) model Kappa statistic.

The first approach, based on AICc, is the conventional approach and is applicable to models of any generalized linear form.

The second approach, based on false positive error rates, is only applicable for binary logistic regression models (i.e., binary, 0 or 1, dependent variable). In this case, at the specified model sensitivity level (default = .95, giving the proportion of true 1's or presences predicted to be 1's or present), alternative models are ranked based on their commission error rate (proportion of true 0's or absences predicted to be 1's or present). The original application for this approach was climate niche envelop modeling for species based on present/absent data, but where the focus is on creating a prediction "envelop" that captures most (e.g., 95%) of the true present observations, which is referred to as model sensitivity. Given this specified high level of model sensitivity, alternative models are evaluated and rank based on the goal of minimizing the errors of commission (i.e., true absences predicted to be present, also referred to as false positives).

The third approach, based on Kappa, is only applicable for binary logistic regression models (i.e., binary, 0 or 1, dependent variable). Kappa is a measure of chance-corrected correct classification rate and is interpreted as the "percentage better than chance". Kappa essentially equally weights both errors of omission and errors of commission, but takes into account the group sample sizes. For each model evaluated in the all subsets, the cutpoint is found that maximizes Kappa. Alternative models are ranked based on their maximum Kappa.

Variable importance, if selected, is evaluated using two different approaches. The first approach assigns variable importance based on the sum of AICc model weights for each variable (i.e., the sum of model weights across all models containing each variable). An important variable is one that is in all the supported models (i.e., models with a deltaAICc <10). Note, the rank and delta arguments not only set the number of models reported in the output tables, but they also determine the models used to compute the variable importance scores. Specifically, variable importance scores are computed as the sum of the AICc model weights for each variable across the models retained in the final list. Thus, even a variable that is in all of the retained models may have a variable importance score << 1 if any of the models not retained have deltaAICc values <10. The second approach is based on the method of hierarchical variance partitioning as implemented in the hier.part library (see help(hier.part) for details of this function and other related functions. NOTE, processing time increases dramatically with increasing number of variables. The variable importance based on hierarchical variance decomposition is not done if the number of variables is greater than 12.

Model averaging, if selected, is done with the MuMIn package and computes both the subset

model-average coefficients and their associated standard errors and confidence intervals in addition to the full model-averaged coefficients. Briefly, the subset averaged coefficients are based on the model-weighted average among only the models including each term, whereas the full model-averaged coefficients are based on the model-weighted average among all the models, including models not containing the term, under the assumption that the coefficient is zero for any model not containing the term.

Value:

all.subsets.glm returns a list containing either one to six components, depending on whether the coef.table, varimp, and model.ave arguments are TRUE or FALSE:

model.statistics: Data frame containing a single row for each model and the following columns:

id:	unique number assigned to each model
model:	list of the independent variables (predictors)
D2:	proportion of deviance explained by the model
AICc:	corrected AIC
deltaAICc:	difference between model AICc and the minimum AICc
wgtAICc:	model AICc weight
cutpoint:	the cutpoint or threshold on the probability of presence scale used to achieve the specified sensitivity level (if select = 'commission').
FP:	false positive (commission error) rate (if select = 'commission')
deltaFP:	difference between model FP and the minimum FP (if select = 'commission')
Kappa:	Kappa statistic (if select = 'Kappa')
deltaKappa:	difference between model Kappa and the minimum Kappa (if select = 'Kappa')
rank:	model rank order based on either AICc, FP, or Kappa

coefficients: [if coef.table=TRUE] Data frame containing a row for each independent variable each time it was in one of the retained models (based on select =) and the following columns:

id:	unique number assigned to each model
Term:	independent variable
Estimate:	parameter estimate from the glm
Std. Error:	standard error of the estimate
z value:	z-statistic
Pr(>|z|):	P-value for the test of the null hypothesis that the parameter estimate is zero

variable.importance.AICc: [if varimp=TRUE] Data frame containing a row for each independent variable in the data set and the following columns. Note, variables included as quadratic terms are combined with their linear terms for purposes of computing variable importance. Thus, a model

that contains both the linear and quadratic terms will get counted once in the computation of AICc variable importance:

Term:	independent variable (including intercept)
N:	number of retained models (based on select = and other arguments such as deltaAIC= and rank=) containing the term
AICc:	variable importance measure based on the sum of AICc model weights across models containing the variable

variable.importance.HP: [if varimp=TRUE] Data frame containing a row for each independent variable in the data set and the following columns. Note, variables included as quadratic terms are treated separately from their linear counterparts for purposes of computing variable importance. Also, variable importance based on hier.part() is only computed if the number of x variables is ≤12:

Term:	independent variable (including intercept)
HP:	variable importance measure given as the percentage of the total variance explained by each independent variable (I.perc value from hier.part())

subset.average.coefficients: [if model.ave=TRUE] Data frame containing a row for each independent variable in the data set and the following columns.

Estimate	subset model-averaged coefficient
Std.Error	standard error of the estimate
Adjjusted SE	adjusted standard error of the estimate
Lower CI	lower 95% confidence interval estimate
Upper CI	upper 95% confidence interval estimate

full.average.coefficients: [if model.ave=TRUE] Data frame containing a row for each independent variable in the data set and the following columns.

Estimate	full model-averaged coefficient

Author:

K. McGarigal, March 29, 2008; modified Sept 26, 2011; Oct 1, 2011; Aug 7, 2013; Dec 26. 2013; Feb 1, 2014; Fe 27, 2014

References:

See help files for hier.part() and MuMIn().

Examples:

```
x<-read.csv('turtle.csv', header=TRUE)
all.subsets.glm(y=x[,2], x1=x[,5:10], family=binomial, maxp=3)
all.subsets.glm(y=x[,2], x2=x[,5:10], x2.linear=TRUE, x.force=x[,11:12], offset=x[,13],
    family=binomial, maxp=5)
all.subsets.glm(y=x[,2], x1=x[,5:10], x2=x[,11,drop=FALSE],x.force=x[,12:14], weights=x[,13],
    family=binomial, maxp=6, select='commission', deltaFP=.2, rank=20, varimp=FALSE,
    model.ave=TRUE)
```

Function: box.plots

Description:

Produces box-and-whisker plots for individual variables (columns) of a data frame. If a grouping variable is specified, box-and-whisker plots for each group are displayed side-by-side on the same page for efficient comparison.

Usage:

box.plots(x, var=' ', by=' ', save.plot=FALSE)

Dependency:

None.

Arguments:

x:	*required* name of data frame containing one or more numeric variables.
var:	*optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.
by:	*optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]
save.plot:	*optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'box.var.jpg', where 'var' is variable name. [default = FALSE]
...	*optional* additional arguments to be passed to the hist and plot functions, including the following defaults: col='blue' color of box and whiskers las=1 horizontal orientation of axis labels

Details:

box.plots is simply a convenience wrapper for the box() function that makes it efficient to quickly produce box-and-whisker plots for many variables, and optionally by a grouping variable. See help(box) for details of box plot construction.

Value:

No object is returned.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
box.plots(x, 'AMGO:WWPE', c('BASIN','SUB'), TRUE)
```

Function: by.names

Description:
Combines two or more grouping variables into a single new grouping variable.

Usage:
by.names(infile, by=names(infile))

Dependency:
None.

Arguments:
infile: *required* name of data frame containing two or more grouping variables. Note, grouping variables can be character or numeric.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. If omitted, all variables in x object are considered as grouping variables. [default = no groups]

Details:
by.names is simply an efficient means of combining two or more grouping variables into a single grouping variable for efficient us in subsequent analyses involving groups. By.names is called by several functions in the mvstats library. The original grouping variables are replaced with two new variables:

id: numeric variable containing a sequential number for unique groups.

groups: character variable containing unique group labels formed by concatenating the names of the levels of each grouping variable. For example, an observation (row) with the value 'A' on the first grouping variable and '2' on the second grouping variable, would get the new value 'A.2'. In addition, the name of the new grouping variable is derived by concatenating the names of the original grouping variables. For example, if the first grouping variable was named 'habitat' and the second grouping variable was named 'stage', the new grouping variable would be named 'habitat.stage'.

Value:
Returns a data frame with the original grouping variables replace by two new variables: id and the combined names of the grouping variables (see details).

Author:
B. Compton, August 30, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
by.names(x, c('BASIN','SUB'))
```

Function: ci.lines

Description:

Adds confidence interval lines to a scatter plot based on a linear regression. Requires an object fit by the lm() function. Typically, following the linear model fit using lm(), a scatterplot of y~x is displayed and the fitted line is added (see example), and then ci.lines() is used to add a confidence envelope, by default a 95% confidence interval, for either the expected value or predicted value around the fitted line.

Usage:

ci.lines(model, interval=95, type='expected')

Dependency:

None.

Arguments:

model: *required* object from lm().

interval: width of confidence interval, given as percent. [default=95]

type: type of confidence interval, either 'expected', or 'predicted' [default='expected'].

Details:

None.

Value:

None. This function does not return an object; it simply calls the lines() function twice to add the upper and lower confidence interval lines to an existing plot. Note, you must produce the scatterplot first before using ci.lines().

Author:

K. McGarigal, November 9, 2016

References:

None.

Examples:

```
xvec<-seq(1:100)
y<-rnorm(100,mean=2+.5*xvec,sd=10)
fit<-lm(y~xvec)
abline(reg=fit)
ci.lines(fit)
```

Function: class.monte

Description:

Monte carlo split-sample cross validation using linear (lda) or quadratic (qda) discriminant analysis. The data set is randomly split into training and validation data sets and for each permutation, the training data set is used to build the classification criterion and then the validation samples are classified. The resulting group-specific and overall correct classification rates (CCR) are computed as well as either the Cohen's Kappa or Tau statistic, which is a chance-corrected measure of classification accuracy. The mean and quantiles of the permutation distribution of each statistic is returned.

Usage:

class.monte(y, grouping, prop=.5, type='qda', perm=1000, priors, ...)

Dependency:

library(MASS)

Arguments:

y:	*required* name of data frame or matrix containing two or more discriminating variables (and only the discriminating variables).
grouping:	*optional* vector (either numeric or character) containing the group membership of each observation. If omitted, then the grouping variable must be in the first column of y. [default = user-specified grouping variable]
prop:	*optional* number specifying the proportion of observations (rows) in y to be used in the 'calibration' data set; the remainder being held out in the 'validation' data set. [default = .5]
type:	*optional* choice of either linear discriminant analysis using the lda() function or quadratic discriminant analysis using the qda() function in the MASS library. [default = qda]
perm:	*optional* number of permutations. [default = 1000]
priors:	*optional* prior probabilities of class membership. If unspecified, the class proportions for the calibration set are used and the Kappa statistic is computed (see details). If specified, the probabilities should be specified in the order of the factor levels and the Tau statistic is computed (see details). [default = proportional to group sample sizes]
...	*optional* additional arguments passed on to the lda() and qda() functions.

Details:

class.monte provides split-sample cross validation of a linear or quadratic discrimination through

repeated random subsetting of the data set into a training and validation set and classifying the hold-out samples in the validation set using the classification criterion derived from the training data set. The classification results are summarized by two statistics: the overall correct classification rate, which is simply the sum of the diagonals of the classification (confusion) matrix divided by the total number of samples classified, and (2) either the Cohen's Kappa statistic or the Tau statistic, which are chance-corrected measures of classification accuracy. The Kappa statistic is the preferred measure when the priors are assumed to be proportional to group sample sizes (see cohen.kappa() for more details). The Tau statistic is an alternative when the priors are known or are not assumed to be equal to sample sizes (see tau() for more details). If the group sample sizes are equal, the two statistics are identical.

In some cases, a random training data set may be ill-conditioned for either lda() or qda() due to having a singular or nearly singular matrix, typically because one of the groups ends up having no variation in one of the discriminating variables or because one of the variables ends up being perfectly predicted by a combination of the remaining variables (i.e., perfect multicolinearity). In such cases, the training data set is deemed "invalid" and skipped. The number of "valid" random training data sets is printed to the console.

Value:

Returns a data frame consisting of a column for the computed correct classification rate (CCR) for each group, plus a 'total' CCR column and a column for either Kappa or Tau, depending on which statistic is computed. Rows correspond to different quantiles of the permutation distribution, including the 0^{th}(minimum), 5^{th}, 50^{th} (median), 95^{th}, and 100^{th}(maximum) percentiles, and the mean of the distribution. The mean total CCR, for example, represents the average overall correct classification rate for the validation data sets.

Author:

K. McGarigal, 4 November 2006; updated 2 February 2013; updated 25 February 2015

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
class.monte(y,grouping=grp,prop=.5,type='qda')
```

Function: clus.composite

Description:

Creates composite clusters by taking the mean of each variable for the sampling entities in each cluster.

Usage:

clus.composite(x, grp)

Dependency:

None.

Arguments:

x: *required* name of data frame containing numeric variables.

grp: *required* vector (character or numeric) of the same length as x giving the cluster membership; typically derived by cutting a dendrogram in a hierarchical clustering using cutree() or as specified in the clustering vector resulting from a nonhierarchical clustering (e.g., using pam, clara, or kmeans).

Details:

clus.composite is a very simple function for calculating the 'mean' vector (i.e., centroid) for each cluster (or group samples) and is typically only appropriate for metric data. For count data, a 'median' composite may be more meaningful, although the median can be quite different from the centroid and thus be deemed undesirable. An alternative for count data is to select the 'medoids'; the sample that "best" represents each cluster, as is produced with the robust k-means clustering in pam() and clara() in the Cluster library.

Value:

Returns a data frame with dimensions k x p, where k equals the number of clusters (groups) and p is the number of variables (columns) in the original data set.

Author:

K. McGarigal, October 18, 2006

References:

None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)]
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.pam<-pam(y.eucl,k=5)
```

```
grp<-y.pam$clustering
y.clus<-clus.composite(y,grp)
```

Function: clus.stats

Description:

Computes a few simple univariate summary statistics (number of observations, mean and coefficient of variation) for each cluster and computes a univariate Kruskall-Wallis rank sum test (nonparametric) of no differences among groups.

Usage:

clus.stats(x, grp)

Dependency:

None.

Arguments:

x: *required* name of data frame containing numeric variables.

grp: *required* vector (character or numeric) of the same length as x giving the cluster membership; typically derived by cutting a dendrogram in a hierarchical clustering using cutree() or as specified in the clustering vector resulting from a nonhierarchical clustering (e.g., using pam, clara, or kmeans).

Details:

clus.stats computes the number of observations in each cluster, and for each variable in x, clus.stats computes the mean and coefficient of variation and conducts a nonparametric test of group differences using the Kruskall-Wallis rank sum test. Other summary statistics can easily be added to the function.

Value:

Returns a list containing three components:

cluster.nobs: a vector containing the number of observations in each cluster.

cluster.mean: a table giving the variables as rows and the mean for each cluster as columns, and the p-value for the Kruskall-Wallis rank sum test in the last column.

cluster.cv: a table as above, but giving the coefficient of variation. A value of NA is given for any cluster containing only a single observation.

Author:

K. McGarigal, October 18, 2006

References:

None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)] #selecting numeric variables of interest
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.pam<-pam(y.eucl,k=5)
grp<-y.pam$clustering
y.stats<-clus.stats(y,grp)
```

Function: cohen.kappa

Description:

Computes Cohen's *Kappa* statistic (Cohen 1960) from a classification table, typically the result of a classification of samples into groups using the predict() function on an object from linear (lda) or quadratic discrimination (qda). Cohen's *Kappa* is a chance-corrected measure of classification accuracy and is suitable when the prior probabilities of group membership are assumed to be proportional to group sample sizes.

Usage:

cohen.kappa(y)

Dependency:

None.

Arguments:

y: *required* name of a table (class = 'table') containing the frequency of observations in each group (column) classified into each group (row). The table must be square and contain a single row and column for each group.

Details:

Kappa is defined as:

$$Kappa = \frac{p_o - \sum_{i=1}^{G} p_i q_i}{1 - \sum_{i=1}^{G} p_i q_i}$$

where G is the number of groups, p_o is the observed percentage of samples correctly classified, p_i is the percentage of samples in the i^{th} group, and q_i is the percentage of samples classified into the i^{th} group.

Kappa is defined here in terms of proportions in keeping with the original development of the statistic, although the terms could easily be defined using frequencies (like *Tau*). Like *Tau*, a *Kappa* of zero indicates no improvement over chance, and a *Kappa* of 1 indicates perfect assignment. An intermediate value of *Kappa* such as 0.82, for example, indicates that classification based on the discriminating variables was 82% better than chance assignment. A *Kappa* that is much lower than the overall correct classification rate suggests that the correct classification rate, and hence group predictability, is inflated and that much of the classification power is due simply to chance.

It is important to note that *Kappa*, like any measure of classification success, is unbiased only when computed with 'holdout' samples. In other words, for unbiased results, the accuracy of the classification criterion should be evaluated by comparing the classification results and chance-

corrected criteria computed from a 'holdout' or 'validation' sample. This is because the classification functions are more accurate for the samples they are derived from than they would be for the full population. Thus, if the samples used in calculating the classification function are the ones being classified, the result will be an upward bias in the correct classification rate.

Value:

Returns an object containing the value of *Kappa*.

Author:

K. McGarigal, November 4, 2006

References:

Cohen, J. 1960. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20: 37-46.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
y.qda<-qda(y,grouping=grp)
y.qda.pred<-predict(y.qda)
y.table<-table(grp,y.qda.pred$class)
cohen.kappa(y.table)
```

Function: col.summary

Description:

Computes a variety of column summary statistics for numeric variables in a data frame, including several statistics appropriate for summarizing a community data set consisting of species abundances.

Usage:

col.summary(x, var=NULL, by=NULL, outfile=NULL, ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize; e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables. [default = all variables]

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). If more than one 'by' variable is specified, then a single grouping variable will be created from the unique combinations of values across variables and the summary statistics will be computed for each group. [default = no groups]

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

Details:

col.summary computes the following column summary statistics for the selected variables (columns), and ignores missing values (na.rm=TRUE), except where noted:

nobs:	number of observations (including missing values)
min:	minimum value
max:	maximum value
mean:	average value
median:	median value (i.e., 50^{th} percentile)
sum:	sum of all values
sd:	sample standard deviation
cv:	coefficient of variation (i.e., 100*sd/mean)
zeros:	number of elements with the value zero
pct.zeros:	percent of observation with the value zero
nobs.missing:	number of missing observations (NA)
pct.missing:	percent of observations with missing values (NA)

se:	standard error (i.e., sd/sqrt(non-missing obs))
se.ratio:	standard error ratio (i.e., 100*se/mean)
richness:	number of non-zero elements

If a 'by' argument is given, these summary statistics are computed for each level of the 'by' variable(s).

Value:

A data frame containing the column summary statistics or, if a 'by' argument is given, a list wherein each component is a data frame containing the column summary statistics for a single group.

Author:

K. McGarigal, February 9, 2008

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
col.summary(x, 'AMGO:BHGR', c('BASIN','SUB'), 'D:/stats/testout')
```

Function: commission.mc

Description:

Conducts a Monte Carlo randomization test of the commission (false positive) error rate (i.e, proportion of true absences that are predicted to be present) given a specified level of model sensitivity (i.e., proportion of true presences predicted to be present). This function works on a fitted generalized linear model (using the glm function) when the dependent variable is binary; i.e., for a binary logistic regression model.

Usage:

commission.mc(fit, test=FALSE, reps=1000, sensitivity=.95, plot=TRUE, ...)

Dependency:

none

Arguments:

fit: *required* fitted glm object from binary logistic regression; i.e., binary dependent variable and binomial error family.

test: *optional* logical (TRUE or FALSE) indicating whether to conduct a Monte Carlo randomization test.. [default=FALSE]

reps: *optional* integer giving the number of Monte Carlo permutations if test=TRUE. [default=1000]

sensitivity: *optional* sensitivity level (proportion of true positives predicted to be positive) for determining the cut point for predicting observations as 0 (absent) or 1 (present); ranges from 0-1 delta; only applicable is select = commission. [default = .95]

plot: *optional* logical (TRUE or FALSE) indicating whether to produce of histogram of the random permutation (i.e., null) distribution of commission error rates along with the original observed commission error rate. [default = TRUE]

... *optional* additional arguments to be passed to the hist() function.

Details:

commission.mc() is a function for conducting a Monte Carlo randomization test of the commission (or false positive) error rate given a specified level of model sensitivity. Specifically, given a glm fitted object from a binary logistic regression, this function extracts the data, the model formula, and any offset and/or weights used in the original fit from the fitted object., computes the false positive (commission) error rate, and then repeatedly permutes the dependent variable (i.e., random shuffle) and recomputes the false positive error rate. The p-value is computed as the proportion of the permutation distribution (i.e, null distribution) that is less than or equal to the observed false positive rate. A histogram of the null distribution of false positive rates and a vertical line showing the observed false positive rate is optionally produced.

Value:

commission.mc returns a data frame containing two or three rows (depending on whether test=TRUE):

Cutpoint: The cutpoint (real number) for classifying observations as 'present' or 'absent'; any observation with a predicted value greater than or equal to the cutpoint is predicted to be 'present', while less than the cutpoint is predicted to be 'absent'. The cutpoint is given on scale of the fitted values, obtained by transforming the linear predictors by the inverse of the link function.

Observed commission rate: False positive (commission) error rate given that the cutpoint is chosen to achieve the specified model sensitivity.

P-value: Proportion of the permuted false positive error rates (representing the null distribution) less than or equal to the observed false positive error rate.

Author:

K. McGarigal, October 2, 2011; January 21, 2014 fixed bug that prevented sensitivity from being anything but the default value of 0.95.

References:

None.

Examples:

```
z<-read.csv('eWothClim.csv', header=TRUE)
fit<-glm(pres~day+ I(day^2) + time + I(time^2) + gdd + I(gdd^2) + pmaysept +
    I(pmaysept^2) + tminjan + tavgjan + I(tavgjul^2), data=z, family=binomial(link=logit),
    offset=effortHrs, weights=w25)
commission.mc(fit1,test=TRUE,reps=1000,sensitivity=.95,plot=TRUE)
```

Function: concordance

Description:

Computes Lin's coefficient of concordance between observed and expected counts in a presence versus available study design; it is intended as a means to validate a resource selection function developed from presence-only data based on the method described in Johnson et al (2006). Specifically, this function takes a binary logistic regression model developed using either standard binary logistic regression or paired (or conditional or case-controlled) logistic regression with or without random effects and optionally conducts a v-fold cross-validation of the model's predictive ability by computing the coefficient of concordance between the (cross-validated) observed counts in the optimized or user-specified number of bins of predicted relative probability of occurrence and the expected counts in the same bins based on a random or exhaustive sample of the study area.

Usage:

```
concordance<-function(use.data,avail.data,formula,method='glm',weights=NULL,
paired=FALSE,use.strata=NULL,avail.strata=NULL,v=10,bins=NULL,
balanced=TRUE,max.p=NULL,obex.plot=TRUE,cc.plot=TRUE,...)
```

Dependency:

library(epiR)
library(lme4) if method= 'glmer' or 'glmer'

Arguments:

use.data: *required* dataframe containing the dependent (response) and independent (predictor) variables for the "present" observations.

avail.data: *required* dataframe containing the dependent (response) and independent (predictor) variables for the "available" observations. Note, the avail.data must have the same exact same columns as the use.data for unpaired designs (i.e., paired=FALSE); additional columns are allowed for paired designs so long as both datasets contain all the predictors in the designated model formula.

formula: *required* model formula created using as.formula().

method: *required* method for fitting the logistic regression model: 'glm' for a fixed effects model, and 'glmer' or 'lmer' [from the lme4 package] for a mixed effects model. [default='glm']

weights: *optional* logical (TRUE or FALSE), or character for specifying the use of weights when fitting the logistic regression model. If NULL, no weights are used. If TRUE, paired must be FALSE, all used points are given a weight of one, and all available points are given a weight such that the sum of the weights of the available points equals the sum of the weights of the used points. If weights is a character, it must the name of the column in use.data and avail.data that contains the weights values.

weights must be a character when paired=TRUE. When paired=FALSE and weights is a character, the user has the option of specifying balanced=TRUE which maintains the equality of sums of weights for used and available points during the cross validation while maintaining the original weights of use.data. If balanced=FALSE, the original weights are maintained during cross-validation. When all used points are given equal weight, setting weights=TRUE will be the simplest approach for maintaining equality of sums of weights during cross-validation. [default=NULL]

paired: *required* logical (TRUE or FALSE) indicating whether the design is "paired", in which case the model is fit with the use.data only, the response variable must be all '1s' and the predictors must be all differences between observed and available, and the model is fit without an intercept. [default = FALSE]

use.strata: *optional* factor containing the stratification for the cross-validation. Note, it must be a "factor" (use as.factor() to coerce if necessary) and it must have the same length and be in the same order as the use.data dataset. If use.strata is NULL, the cross-validation is based on random subsets of the data. [default = NULL]

avail.strata: *optional* factor containing the stratification of the available data for the cross-validation. Note, it must be a "factor" (use as.factor() to coerce if necessary) and it must have the same length and be in the same order as the avail.data dataset. In a paired design (paired=TRUE), if use.strata is specified and avail.strata is NULL, the cross-validation is based on the strata in the use.data and the entire avail.data is used to validate each fold of the cross-validation. On the other had, if both the use.strata and avail.strata are specified, they must contain the same strata. In an unpaired design (paired=FALSE), if use.strata is specified then avail.strata must also be specified and they must contain the same strata. [default = NULL]

v: *optional* integer specifying the number of random folds for the cross-validation. Note, this argument is ignored if use.strata is specified. In addition, if v=1, then the full dataset is used to both train the model and valid it (i.e., resubstitution validation); otherwise a cross-validation is conducted using v folds. [default = 10]

bins: *optional* integer specifying the number of bins to use for computing the coefficient of concordance. If bins is NULL, the number of bins is optimized between 5-20; i.e., the number of bins that maximizes the coefficient of concordance is selected. [default = NULL]

balanced: *optional* logical (TRUE or FALSE) indicating whether or not to have sum of weights of use.data equal the sum of weights of avail.data during cross-validation. Only used if weights is a character specifying the column name of the weights value in both use.data and avail.data. [default=TRUE]

max.p: *optional* numeric to specify the maximum allowable relative probability value when

creating the bins. If relative probability values range from 0-1 then max.p is automatically set to one but in cases where the number of random points far exceeds the number of used points the maximum relative probability value may be much less than one. Adjusting max.p allows the bins to evenly span the range of allowable relative probability values when calculating the coefficient of concordance. [default=NULL]

obsex.plot: *required* logical (TRUE or FALSE) indicating whether to create a plot of the observed versus expected proportions with a perfect diagonal reference line (i.e., slope=1) through the origin (i.e., intercept=0) depicted as the reference for perfect concordance. [default = TRUE]

cc.plot: *required* logical (TRUE or FALSE) indicating whether to create a plot of concordance against the number of bins when the number of bins is being optimized (i.e., bins=NULL). [default = TRUE]

... *optional* additional graphics arguments to be passed to the plot() function.

Details:

concordance() is a flexible function for validating resource selection functions based on a binary logistic regression of present-only data. The function has several key features:

- The logistic regression can be based on an *unpaired* design, in which case the response data must be coded 0 for the available or pseudo-absence data (avail.data) and 1 for the present or use data (use.data), or it can be based on a *paired* design, in which case the response data must be coded all 1's and the predictors are given as differences between observed and the conditionally (case-controlled) available for each observation (Agresti 2002).
- The logistic regression can be based on *fixed effects* only, in which case the model is fit with glm(), or it can be based on *mixed effects*, in which case the model is fit with glmer() or lmer() from the lme4 package.
- The optional cross-validation can be based on a user-specified number of random folds (subsets), or it can be based on a user-provided stratification variable.
- The number of predicted probability bins for computing the coefficient of concordance can be optimized over the range 5-20, or it can be user-specified.

Value:

concordance()returns a list containing three major components:

coefficients: Data frame containing the estimated coefficients for the fixed effects in the model for each fold of the data. If cross-validation is not selected, the data frame contains a single row with the estimated model coefficients. If cross-validation is selected, each row represents the estimated coefficients for one fold of the cross-validation.

standard.errors: Data frame containing the estimated standard errors of the coefficients for the fixed effects in the model for each fold of the data. If cross-validation is not selected, the data frame contains a single row with the estimated standard errors. If cross-validation is selected, each row represents the estimated

standard errors for one fold of the cross-validation.

coefficient.concordance: List containing several components regarding the computation of concordance, including:
- v-folds: number of v-folds specified or determined by the stratification variable specified
- number of bins: number of bins used to compute the maximum coefficient of concordance
- bin midpoint probabilities: midpoints of the bins
- observed counts: number of 'present' observations in each bin
- observed proportions: proportion of observed counts in each bin
- available counts: number of 'available' observations in each bin
- expected counts: expected number of observations in each bin
- expected proportions: proportion of expected counts in in each bin
- coefficient of concordance: estimated coefficient of concordance and 95-percent confidence interval

Author:

K. McGarigal, January 24, 2014

References:

Agresti A. (2012). Categorical Data Analysis, 3rd Edition. John Wiley and Sons. 744 pp.

Johnson, CJ, SE Nielsen, EH Merrill, TL McDonald and MS Boyce. 2006. Resource selection functions based on use-availability data: theoretical motivation and evaluation methods. The Journal of Wildlife Management 70:347-357.

Examples:

```
data<-read.csv('c:/work/papers/murrelets/KIMUdata.csv',header=TRUE)
use.data<-data[data$presence_absence==1,]
avail.data<-data[data$presence_absence==0,]
formula.glm<-as.formula(presence_absence~depth+dist.shore+dist.stream+dist.glacier+dist.morain
e)
concordance(use.data=use.data,avail.data=avail.data, formula=formula.glm,n.vars=5)
concordance(use.data=use.data,avail.data=avail.data,formula=formula.glm,n.vars=5,v=10,bins=10)
concordance(use.data=use.data,avail.data=avail.data,formula=formula.glm,use.strata=use.data$year,
avail.strata=avail.data$year, n.vars=5)
concordance(use.data=use.data,avail.data=avail.data,formula=formula.glm,use.strata=use.data$year,
avail.strata=avail.data$year, n.vars=5,bins=10)
```

Function: contrast.matrix

Description:

Creates an indicator distance matrix (i.e., class = 'dist') defining the contrast between groups. Specifically, given a vector of length N, either character (factor) or numeric and defining the group membership of each observation, this function creates an N by N distance matrix (with class = 'dist') where the elements are 0 if the pairwise samples are in the same group and 1 if they are from different groups. This indicator matrix is intended to be used as the X-matrix in a Mantel test, where the test is for significant differences among groups.

Usage:

contrast.matrix(grp)

Dependency:

None.

Arguments:

grp: *required* vector (character or numeric) that defines the group membership of each observation; for example, as might be extracted from a data frame containing a variable (column) that defines group membership, or as might be derived by cutting a dendrogram in a hierarchical clustering using cutree(), or as specified in the clustering vector resulting from a nonhierarchical clustering (e.g., using pam, clara, or kmeans).

Details:

Regardless of the number of groups specified in the grp vector, contrast.matrix() will always return a distance matrix containing just 0's (pairwise samples within the same group) and 1's (pairwise samples from different groups). This is the appropriate structure for the Mantel test of no group differences.

Value:

Returns an N by N distance matrix (class = 'dist').

Author:

K. McGarigal, October 25, 2006

References:

None.

Examples:

```
#extract from data frame a variable (column) containing group membership
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]

#extract group membership vector by cutting a dendrogram from hierarchical clustering
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
```

```
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)]
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.eucl.ward<-hclust(y.eucl,method='ward')
plot(y.eucl.ward,main='Wards-linkage Dendrogram',xlab='Species',labels=x[,1])
rect.hclust(y.eucl.ward,k=5) #cut by #clusters
grp<-cutree(y.eucl.ward,5)
contrast<-contrast.matrix(grp)
```

Function: cov.test

Description:

Simply a convenience wrapper for the bartlett.test() and fligner.test() functions to simplify testing multiple variables in a data set. Performs a unvivariate test of homogeneity of variance among groups using either Bartlett's test for k samples (i.e., two or more groups) or Fligner's test for a rank-based (nonparametric) k-sample test for homogeneity of variances. These are both tests of the null hypothesis that the variances in each of the groups (samples) are the same.

Usage:

cov.test(x, groups, var=' ', method='bartlett', ...)

Dependency:

None.

Arguments:

x:	*required* numeric vector of data values, or data frame or matrix containing one or more numeric variables, or a list of numeric data vectors representing the respective samples, or fitted linear model objects (inheriting from class '"lm"').
grouping:	*required* vector (either numeric or character) containing the group membership of each observation in x.
var:	*optional* list of variables in x to test. If omitted, all variables in x are tested. [default = test all variables]
method:	*optional* choice of either Bartlett's parametric test or Fligner's nonparametric test for homogeneity of variances. [default = 'bartlett']
...	*optional* additional arguments passed on to the bartlett.test() and fligner.test() functions.

Details:

None.

Value:

Returns a data frame containing three columns. The first column lists the variable name, the second column gives the test statistic (either 'bartlett' or 'fligner'), and the third column gives the p-value for the null hypothesis test. There is a separate row for each variable tested.

Author:

K. McGarigal, November 4, 2006

References:

See bartlett.test() and fligner.test().

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
cov.test(y,grp,method='fligner')
```

Function: data.dist

Description:

Calculates a dissimilarity/distance matrix from a numeric data frame or matrix based on any number of different dissimilarity/distance measures. Data.dist is mostly a convenience wrapper for the vegdist() function in the vegan library (see details), but adds correlation distance and mahalanobis distance to the list of distance measures.

Usage:

data.dist(x, method, var=' ', cor.method='pearson', abs=FALSE, outfile=' ', binary=FALSE, diag=FALSE, upper=FALSE, na.rm=TRUE, ...)

Dependency:

library(vegan)

Arguments:

x: *required* name of data frame or matrix containing one or more numeric variables.

method: *required* name of dissimilarity/distance measure: 'manhattan', 'euclidean', 'correlation', 'mahalanobis', 'bray', 'kulczynski', 'jaccard', 'gower', 'morisita', 'horn', 'mountford', 'raup', or 'binomial' (see details below).

var: *optional* list of one or more numeric variables to use in the distance calculation, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

cor.method: *optional* choice of correlation measure: 'pearson', 'spearman', or 'kendall' (see details below). [default = 'pearson']

abs: *optional* logical (TRUE or FALSE) when method=correlation indicating whether distance should be based on the absolute value of the correlation coefficient (see details below). [default = FALSE]

outfile: *optional* name of an output file containing the distance matrix in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

binary: *optional* logical (TRUE or FALSE) to convert the data to binary presence/absence before computing distance. [default = FALSE]

diag: *optional* logical (TRUE or FALSE) to return the diagonals of the distance matrix to the console. Note, the diagonals are always printed in the saved distance matrix if outfile is specified. [default = FALSE]

upper: *optional* logical (TRUE or FALSE) to return the upper half of the distance matrix to

the console. Note, the upper half is always stored in the saved distance matrix if outfile is specified. [default = FALSE]

na.rm: *optional* logical (TRUE or FALSE) whether to use pairwise deletion of missing values when computing distances. [default=TRUE]

... *optional* other parameters for method = 'gower' which accepts 'range.global' parameter of 'decostand'.

Details:

As noted above, data.dist is mostly a convenience wrapper for the vegdist() function in the vegan library that simply adds the correlation and mahalanobis distance metrics. See help(vegdist) for details and references; but, briefly, the following distance measures are available:

euclidean: d[jk] = sqrt(sum (x[ij]-x[ik])^2)

manhattan: d[jk] = sum(abs(x[ij] - x[ik]))

correlation: if abs=FALSE (default):

d[jk] = (1 - r[ij])/2

where r[ij] is the correlation coefficient between samples j and k. d[jk] ranges from 0 to 1, where 0 equals a correlation of 1 and 1 equals a correlation of -1 and .5 equals a correlation of 0.

else if abs=TRUE:

d[jk] = abs(r[ij])

where r[ij] is the correlation coefficient between samples j and k. d[jk] ranges from 0 to 1, where 0 equals a correlation of 1 or -1 and 1 equals a correlation of 0.

The correlation coefficient is specified in the cor.method and can be one of the following options:

pearson: Pearson product-moment correlation
kendall: Kendall' tau rank-based measure of association
spearman: Spearman's rho rank-based measure of association

mahalanobis: $d[jk] = \text{sqrt}((X[j] - X[k])' \cdot S^{-1} \cdot (X[j] - X[k]))$

where X[j] - X[k] = vector of differences between sites j and k (across all p variables) and S^{-1} = inverse of the covariance matrix of the dataset X.

Note, this returns mahalanobis distance rather than the mahalanobis distance squared (which is commonly reported). In addition, note that this distance metric may only be only meaningful if the number of observations is much greater than the number of variables; otherwise, it is possible for the mahalanobis distance to be constant across all pairwise comparisons.

<u>proportional city-block measures</u>:

gower: d[jk] = sum (abs(x[ij]-x[ik])/(max(x[i])-min(x[i]))

canberra: d[jk] = (1/NZ) sum ((x[ij]-x[ik])/(x[ij]+x[ik])),
where NZ is the number of non-zero entries.

bray: d[jk] = (sum abs(x[ij]-x[ik])/(sum (x[ij]+x[ik]))

Jaccard: 2B/(1+B), where B is Bray-Curtis ('bray') dissimilarity.

kulczynski: d[jk] = 1 - 0.5*((sum min(x[ij],x[ik])/(sum x[ij]) + (sum min(x[ij],x[ik])/(sum x[ik]))

<u>Other distance measures</u>:

morisita: d[jk] = 2*sum(x[ij]*x[ik])/((lambda[j]+lambda[k]) * sum(x[ij])*sum(x[ik])),
where lambda[j] = sum(x[ij]*(x[ij]-1))/sum(x[ij])*sum(x[ij]-1)

horn: Like 'morisita', but lambda[j] = sum(x[ij]^2)/(sum(x[ij])^2)

binomial: d[jk] = sum(x[ij]*log(x[ij]/n[i]) + x[ik]*log(x[ik]/n[i]) - n[i]*log(1/2))/n[i]
where n[i] = x[ij] + x[ik]

Binomial index is derived from Binomial deviance under null hypothesis that the two compared communities are equal. It should be able to handle variable sample sizes. The index does not have a fixed upper limit, but can vary among sites with no shared species. For further discussion, see Anderson & Millar (2004).

mountford: M = 1/alpha where alpha is the parameter of Fisher's log series 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(a*M) + exp(b*M) = 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). The Mountford index is in the range 0... log(2), but the dissimilarities are divided by log(2) so that the results will be in the conventional range 0 ... 1.

raup: Raup-Crick dissimilarity 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. 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.

Value:

A distance matrix.

Author:

K. McGarigal, Updated February 14, 2013

References:

See references for vegdist in the vegan library.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
data.stand(x, 'range', 'AMGO:WWPE', 'column', 'D:/stats/testout', TRUE, FALSE, TRUE)
```

Function: data.stand

Description:

Performs various data standardizations for numeric data in a data frame, including both row and/or column standardizations. Data.stand is largely a convenience wrapper for the decostand() function in the vegan library (see details). Note, in contrast to data transformations (see data.trans), standardizations (or relativizations) adjust matrix elements by a row or column standard (e.g., maximum, sum, mean, etc.). In addition, data.stand produces paired histograms depicting the raw (unstandardized) data paired against the standardized data, both with kernel density overlays, for efficient evaluation of the effects of the standardization on the distribution of each variable (column).

Usage:

data.stand(x, method, var=' ', margin='column', outfile=' ', plot=TRUE, save.plot=FALSE, na.rm=TRUE, ...)

Dependency:

library(vegan)

Arguments:

x: *required* name of data frame containing one or more numeric variables.

method: *required* data standardization method: 'total', 'max', 'freq', 'normalize', 'range', 'standardize', 'chi.square', 'hellinger', or 'wisconsin' (see details below).

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

margin: *optional* choice of 'column' or 'row' margins for the standardization. Each method has a default margin (see details), but this can be overridden here. If method = 'wisconsin', margin is ignored, since this is a special double standardization involving both row and column standards. [default = 'column']

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

plot: optional logical (TRUE or FALSE) to determine whether paired histograms depicting the raw and standardized data are plotted or not. [default = TRUE]

save.plot: *optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'shist.var.jpg', where 'var' is variable name. [default = FALSE]

na.rm: *optional* logical (TRUE or FALSE) whether to ignore missing values. [default=TRUE]

... *optional* additional arguments to be passed to the decostand, hist() and plot() functions, including the following graphical defaults:
col.hist='blue' color of histogram
col.line='black' color of the kernel density line
las=1 horizontal orientation of axis labels
lab=c(5,5,4) number of x and y tick marks and length of labels

Details:

It is often desirable to standardize (or relativize) ecological data. Sometimes this is done to place variables, measured in different units and scales, on equal footing. In community data sets consisting of species abundance data, it is often desirable to standardize the data to shift the focus from absolute abundance profiles to relative abundance profiles, either to eliminate or reduce the effect of differences in species abundances or differences in plot totals. The consequences of standardization can be dramatic or subtle depending on the heterogeneity of the data. For example, a row total standardization (i.e., adjusting each sample to a relative abundance profile, so that the adjusted row totals are all equal to 1) can have a profound effect on the results of subsequent analyses if the rows differ greatly in total abundance. But if they differ only slightly, then a row total standardization will have little effect. In any case, standardization is an extremely important tool. As noted above, data.stand is simply a convenience wrapper for the decostand() function in the vegan library. See the help(decostand) for details and references; but, briefly, the following transformations are available:

total: divide by margin total. [default margin = 'row']

max: divide by margin maximum. [default margin = 'column']

freq: divide by margin maximum and multiply by number of nonzero items, so that the average of non-zero entries is one (Oksanen 1983). [default margin = 'column']

normalize: make margin sum of squares (i.e., sum(x^2))equal to one (note, this is not the sums of squared deviation, or sums of squares for shorthand). [default margin = 'row']

range: standardize values into range 0 ... 1. If all values are constant, they will be transformed to 0. [default margin = 'column']

standardize: scale into zero mean and unit variance. [default margin = 'column']

hellinger: square root of method = 'total' [default margin = 'row']

chi.square: divide by row sums (i.e., row totals) and square root of column sums (i.e., column totals), and adjust for square root of matrix total. When used with Euclidean distance, the matrix should be similar to the the Chi-square distance used in correspondence analysis.

wisconsin: divide by row maximum (i.e., row max standardization) followed by column totals (i.e., column total standardization). This is a common 'double' standardization that first adjusts sample differences through a rather drastic row maximum standardization and then adjusts species differences through the less drastic column total standardization. While this double standardization has proven useful in ordination of community sets, the double standardization makes interpretation of the standardized data values exceedingly difficult.

Note, most of these standardizations have been employed for community data sets consisting of species abundances. However, these standardizations may also be useful for other types of ecological data, although in these cases additional caution is warranted before applying these standardizations. In addition, note that most standardization methods will give non-sense results with negative data entries that normally should not occur in the community data. If there are empty sites or species (or constant with 'method = 'range'), many standardizations will change these into 'NaN'.

Value:

A new *data frame* with the standardized data.

Author:

K. McGarigal, September 14, 2006

References:

See references for decostand in the vegan library.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
data.stand(x, 'range', 'AMGO:WWPE', 'column', 'D:/stats/testout', TRUE, FALSE, TRUE)
```

Function: data.trans

Description:

Performs various monotonic data transformations for numeric data in a data frame, including the natural log, power, logit, arcsin square root, and cosine transformations. Note, these transformations are applied to each element of the data matrix, independent of the other elements. They are 'monotonic' because they change the values of the data points without changing their rank. In addition, data.trans produces paired histograms depicting the raw (untransformed) data paired against the transformed data, both with kernel density overlays, for efficient evaluation of the effects of the transformation on the distribution of each variable (column).

Usage:

data.trans(x, method, var=' ', exp=1, ref=0,outfile=' ', plot=TRUE, save.plot=FALSE,...)

Dependency:

None.

Arguments:

x:	*required* name of data frame containing one or more numeric variables.
method:	*required* data transformation method: 'log','power', 'logit', 'arcsin', or 'cosine' (see details below).
var:	*optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.
exp:	*optional* exponent in the range 0-1 for the power transformation (method = 'power'). Note, exp=0 results in a binary (presence/absence) transformation; exp=.5 is equivalent to the square root transformation; and exp=1 results in no change. [default = 1]
ref:	*optional* reference axis in the range 0-360 for the cosine transformation (method = 'cosine'). Note, ref=0 specifies a north-south reference axis such that northerly aspects tend toward 1 and southerly aspects tend toward 0 in the result. The ref= aspect is used to specify a reference axis that differs from north-south. For example, use ref=45 to specify a northeast-southwest reference axis. [default = 0]
outfile:	*optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]
plot:	optional logical (TRUE or FALSE) to determine whether paired histograms depicting the raw and transformed data are plotted or not. [default = TRUE]

save.plot: *optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'thist.var.jpg', where 'var' is variable name. [default = FALSE]

... *optional* additional arguments to be passed to the qqnorm() and qqline() functions, including the following defaults:
col.hist='blue' color of histogram
col.line='black' color of the kernel density line
las=1 horizontal orientation of axis labels
lab=c(5,5,4) number of x and y tick marks and length of labels

Details:

It is often necessary or desirable to transform the raw numerical data to improve the distribution of the data. Ecological data are commonly highly skewed and may extend over two or more orders of magnitude because of a few extreme values. This is especially common in community data sets comprised of count data and biomass data. It is usually desirable to reduce the absolute range of variation in these cases to reduce the effect of these extreme values. The log and power transformations are useful in these situations. In addition, it is sometimes useful to binarize species data to examine the presence/absence signature separate from the quantitative abundance signature, or in some cases there is insufficient quantitative variation to warrant an analysis of quantitative abundances. The zero-power transformation is a quick and easy way to binarize the data into presence/absence.

log: The natural logarithmic transformation is valid for positive data (i.e., x>0) and results in unbounded values (i.e., negative infinity to positive infinity). The log transformation compresses high values and spreads low values by expressing the values as orders of magnitude. Because the log of zero is undefined, a small constant is typically added to all data points. If the lowest nonzero value in the data is one (as in count data), then it is best to add one to each data point before the transformation, because the log(1) = 0. If the lowest nonzero value differs from one by more than an order of magnitude, then adding one will distort the relationship between zeros and other values in the data set. To address this, the generalized procedure outlined in McCune and Grace (2002) is followed that (1) tends to preserve the original order of magnitudes in the data and (2) results in values of zero when the initial value was zero. Given:

min(x) = the smallest nonzero value in the data
int(x) = truncates x to an integer by dropping digits after the decimal point
c = order of magnitude constant = int(log(min(x))
d = decimal constant = $\log^{-1}(c)$
then the log transformation is given as: $x_{ij}^{*} = \log(x_{ij} + d) - c$

power: The power transformation is given as: $x_{ij}^{*} = x_{ij}^{p}$

The zero-power transformation (i.e., p=0) is valid for data in any range and results in binary data (0's and 1's) . All other power transformations (i.e., p>0) are valid for non-negative data (i.e., x≥0) and result in non-negative data. Different exponents change the effect of the transformation. p=0.5 gives the square root transformation, which is similar

in effect to, but less drastic than, the log transformation. The smaller the exponent, the more the compression applied to high values. Exponents less than say 0.3 are essentially presence-absence transformations, yielding values close to 1 for all nonzero values.

logit: The logit transformation is valid only for proportion data, 0-1 inclusive, and results in unbounded data. It is given as: $x_{ij}^{*} = \log\left(x_{ij} / 1 - x_{ij}\right)$

The logit transformation spreads the ends of the scale for proportion data, while compressing the middle. This transformation is generally recommended for proportion data, often improving normality, when used in regression.

asin: The arcsine-squareroot transformation is valid only for proportion data, 0-1 inclusive, and results in data in the same range. It is given as: $x_{ij}^{*} = 2/pi \cdot \arcsin\left(\sqrt{x_{ij}}\right)$

Like the logit transformation, the arcsine-squareroot transformation also spreads the ends of the scale for proportion data, while compressing the middle, but is slightly "weaker" in its affect than the logit transformation. This transformation is often recommended for proportion data when used in multivariate statistical analyses. Note, the multiplication of 2/pi simply rescales the result to range from 0 to 1.

cosine: The cosine transformation is valid only for circular data given in degrees, 0-360, inclusive, and results in data ranging 0-1, with 1 representing the specified reference angle (z = 0 degrees by default) and 0 representing 180 degrees from the reference angle. It is given as: $x_{ij}^{*} = \left(\cos\left(\mathrm{radians}\left(z - x_{ij}\right)\right) + 1\right) / 2$

The cosine transformation linearizes circular data along a reference axis. The default reference axis is north (0 degrees) - south (180 degrees), but can be changed using the ref= argument. For example, to specify a northeast-southwest reference axis, simply specify ref=45. Note, this transformation is essential for any circular variable used as an independent variable in the statistical model. When the circular variable is used as the dependent variable, no transformation is necessary as long as circular statistical methods are employed that utilize an appropriate circular probability distribution such as the Von Mise or Wrapped Cauchy (at least in the context of parametric modeling).

Value:

A new *data frame* with the transformed data.

Author:

K. McGarigal, September 14, 2006

References:

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

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
```

```
data.trans(x, 'power', 'AMGO:WWPE', 0, 'D:/stats/testout', TRUE, FALSE)
```

Function: dist.plots

Description:

Produces three distribution plots of the dissimilarities in a dissimilarity/distance matrix, including a box-and-whisker plot of the between- and within-group dissimilarities, a histogram of the within-group dissimilarities in each group, and a normal quantile-quantile plot for the within-group dissimilarities in each group. This function is simply a convenience wrapper for the boxplot(), hist() and qqnorm() functions that makes it efficient to quickly produce grouped box-and-whisker plots, histograms and qqnorm plots from a dissimilarity matrix. These plots can be useful for assessing the assumptions of various parametric and nonparametric tests of group differences.

Usage:

dist.plots(x, groups, distance='euclidean', na.rm=TRUE, ...)

Dependency:

library(vegan) if x is data frame or matrix.

Arguments:

x: *required* name of data frame or matrix containing one or more numeric variables or a distance matrix (i.e., class = 'dist') typically derived from a function like vegdist(), dist() or data.dist().

groups: *required* vector (either numeric or character) containing the group membership of each observation in x.

distance: *optional* name of dissimilarity/distance measure for use in the vegdist() function in the vegan library: 'manhattan', 'euclidean', 'euclidean', 'bray', 'kulczynski', 'jaccard', 'gower', 'morisita', 'horn', 'mountford', 'raup', or 'binomial' (see vegdist() for details). [default = 'euclidean']

na.rm: *optional* logical (TRUE or FALSE) for pairwise deletion of missing observations when computing dissimilarities. [default=TRUE]

... *optional* additional arguments to be passed to the plotting functions, including the following defaults:

col='blue'	color of box and whiskers, histograms and qqnorm points
col.line='red'	color of line for kernel density overlay and qqline
las=1	horizontal orientation of axis labels

Details:

dist.plots produces three different types of plots useful for examining the distribution of among- and within-group dissimilarity values in a dissimilarity matrix. See Value below for a description of each plot.

Value:

Returns three different plots:

Plot 1: The first plot is a grouped *box-and-whisker plot* of the among- and within-group dissimilarities. The first box-and-whisker represents the distribution of the among- or between-group dissimilarities. The remaining boxes represent the distribution of the within-group dissimilarities for each group.plot.

Plot 2: The second plot is a set of *histograms* of the within-group dissimilarities for each group with a kernel density line overlay.

Plot 3: The third plot is set of *normal quantile-quantile* plots for the within-group dissimilarities for each group.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
dist.plots(y.eucl,grp)
```

Function: distributions

Description:

Suite of plotting functions of two types for examining several different common probability distributions. The first plotting function (e.g., dbinom.plot) produces a plot of the *probability mass function* (pmf) for discrete distributions or a *probability density function* (pdf) for continuous distributions for any specified combination of values for the parameters of the distribution. A single plot is produced with a separate series or curve for each unique combination of parameter values. The second plotting function (e.g., binom.plot) produces a plot containing four subplots: (1) pmf or pdf, as above, (2) *cumulative mass function* (cmf) for discrete distributions or *cumulative density function* (cdf) for continuous distributions, (3) *quantile mass function* (qmf) for discrete distributions or *quantile distribution function* (qdf) for continuous distributions, and (4) a histogram of a random sample drawn from the specified distribution. A separate plot is produced for each unique combination of parameter values. These functions are simply convenience wrappers for the built-in R functions for some of the common probability distributions.

Usage:

Note, in the examples below, the parameters values are simply examples, but can be substituted for any values.

Discrete distributions:

Binomial distribution plots:

```
dbinom.plot(size=10,prob=c(.1,.5,.9))
binom.plot(size=c(10,100),prob=c(.1,.5,.9))
```

Poisson distribution plots:

```
dpois.plot(events=25,lambda=c(.5,1,3,12))
pois.plot(events=25,lambda=c(.5,1,3,12))
```

Negative binomial plots:

```
dnbinom.plot(events=25,mu=c(1,2),size=c(.1,1,10))
nbinom.plot(events=25,mu=c(1,2),size=c(.1,1,10))
```

Geometric distribution plots:

```
dgeom.plot(events=25,prob=c(.2,.5,.7))
geom.plot(events=25,prob=c(.2,.5,.7))
```

Continuous distributions:

Normal distribution plots:

```
dnorm.plot(mean=c(10,12),sd=c(1,2,3),xlim=c(0,20),ylim=c(0,1))
norm.plot(mean=c(10,12),sd=c(1,2,3),xlim=c(0,20))
```

Gamma distribution plots:
dgamma.plot(shape=c(1,2,5),scale=c(1,2,3),xlim=c(0,25),ylim=c(0,1))
gamma.plot(shape=c(1,2,5),scale=c(1,2,3),xlim=c(0,25))

Exponential distribution plots:
dexp.plot(rate=c(1,.5,.1),xlim=c(0,15),ylim=c(0,1))
exp.plot(rate=c(1,.5,.1),xlim=c(0,15))

Beta distribution plots:
dbeta.plot(shape1=c(.5,1,2,5),shape2=c(.5,1,2,5),ylim=c(0,5))
beta.plot(shape1=c(.5,1,2,5),shape2=c(.5,1,2,5))

Lognormal distribution plots
dlnorm.plot(mean=c(0,2),sd=c(.2,.5,1),xlim=c(0,15),ylim=c(0,2))
lnorm.plot(mean=c(0,2),sd=c(.2,.5,1))

Chi-square distribution plots
dchisq.plot(df=c(1,10,100))
chisq.plot(df=c(1,10,100))

Fisher's *F* distribution plots
df.plot(df1=c(1,2,20),df2=c(10,20))
f.plot(df1=c(1,2,20),df2=c(10,20))

Student's *t* distribution plots
dt.plot(df=c(1,10,100))
t.plot(df=c(1,10))

Dependency:
None.

Arguments:

param1: *optional* value(s) for the first parameter of the corresponding distribution. For example, the first named parameter of the binomial distribution is 'size'. The syntax for specifying multiple values is "=c(value1, value2, ...)" or "=c(value1:value2)". Note, some functions will only accept a single value for the first parameter. [default = arbitrary number]

param2: *optional* values(s) same as param1.

param3: *optional* values(s) same as param1.

xlim: *optional* values in some of the functions to control the range of values for the x-axis. The range must be appropriate given the specified parameter values. [default = arbitrary range]

ylim: *optional* values in some of the functions to control the range of values for the y-axis. The range must be appropriate given the specified parameter values. [default = arbitrary range]

Details:
See the corresponding help files in R for each of the distributions; e.g. ?dbinom.

Value:
Returns plots as described above.

Author:
K. McGarigal, February 28, 2009

References:
None.

Examples:
see usage above.

Function: drop.var

Description:

Drops variables (columns) from a data frame based on specified thresholds in any combination of three column summary statistics: coefficient of variation, percentage of zeros and percentage of missing values.

Usage:

drop.var(x, var=' ', outfile=' ', min.cv=0, min.po=0, min.fo=0, max.po=100, max.fo=nrow(x), pct.missing=100)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

min.cv: *optional* threshold level for the column coefficient of variation (cv=100*sd/mean); variables *below* this level will be dropped. [default = 0]

min.po: *optional* minimum percent occurrence; i.e., minimum percentage of samples (rows) with a non-zero value. Variables (columns) with a percent occurrence below this level will be dropped. This is generally used to drop rare species in community data sets. [default = 0]

min.fo: *optional* minimum frequency of occurrence; i.e., minimum number of samples (rows) with a non-zero value. Variables (columns) with a frequency of occurrence below this level will be dropped. This is generally used as an alternative to 'minpo' to drop rare species in community data sets. [default = 0]

max.po: *optional* maximum percent occurrence; i.e., maximum percentage of samples (rows) with a non-zero value. Variables (columns) with a percent occurrence above this level will be dropped. This is generally used to drop ubiquitous species (i.e., present everywhere) in community data sets. [default = 0]

max.fo: *optional* maximum frequency of occurrence; i.e., maximum number of samples (rows) with a non-zero value. Variables (columns) with a frequency of occurrence above

this level will be dropped. This is generally used as an alternative to 'maxpo' to drop ubiquitous species (i.e, present everywhere) in community data sets. [default = 0]

pct.missing: *optional* threhold level for the column percentage of missing values; variables *above* this level will be dropped. [default =100].

Details:

Insufficiently sampled variables can have a deleterious effect on many multivariate analyes. Variables with too little information may not have deleterious effect, but may inconsequential as they offer little in terms of pattern of variation. More importantly, variables with too many missing values may be poorly sampled and may result in too many observations being deleted or ignored in subsequent analyses due to incomplete observations. Finally, variables with too few non-zero elements may not provide sufficient information to reliable estimate a response profile. For example, it is inappropriate to try an estimate a species optimum along an environmental gradient with only 2 or 3 non-zero occurrences. In this case, *sufficiency* is the extent to which each species' ecological character is accurately described by the data. Species with very few records are not likely to be accurately placed in ecological space. You must decide at what level of frequency of occurrence you want to accept the 'message' and eliminate species below this level.

Value:

A new *data frame* without the offending variables.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
drop.var(x, 'AMGO:WWPE', 'D:/stats/testout', min.cv=5, max.po=95, pct.missing=10)
```

Function: edf.plots

Description:

Produces empirical distribution function (EDF) plots for individual variables (columns) of a data frame. If a grouping variable is specified, EDF plots for each group are displayed in panels on the same page for efficient comparison.

Usage:

ecdf.plots(x, var=' ', by=' ', ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]

... *optional* additional arguments to be passed to the plot() and line() functions, including the following defaults:

col='blue'	color of the edf
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

edf.plots is simply a convenience wrapper for the plot() and line() functions that makes it efficient to quickly produce EDF plots for many variables, and optionally by a grouping variable. See help(plot) and help(line) for details on those functions. Note, the EDF is simply a plot of the observed values on the y-axis in rank order from smallest to largest.

Value:

No object is returned.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv(‘testbird.csv’, header=TRUE)
edf.plots(x, 'AMGO:WWPE', c('BASIN','SUB'))
```

Function: ecdf.plots

Description:

Produces empirical *cumulative* distribution function (ECDF) plots for individual variables (columns) of a data frame. If a grouping variable is specified, ECDF plots for each group are displayed in panels on the same page for efficient comparison.

Usage:

ecdf.plots(x, var=' ', by=' ', ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]

... *optional* additional arguments to be passed to the plot() and ecdf() functions, including the following defaults:

col='blue'	color of the ecdf
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

ecdf.plots is simply a convenience wrapper for the ecdf() function that makes it efficient to quickly produce ECDF plots for many variables, and optionally by a grouping variable. See help(plot) and help(ecdf) for details on those functions. Note, the ECDF gives the probability, or proportion, of values (on the y-axis) falling below any given value of x. Compare this to the raw empirical distribution (EDF) which is simply a plot of the observed values of the variable in rank order from smallest to largest.

Value:

No object is returned.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
ecdf.plots(x, 'AMGO:WWPE', c('BASIN','SUB'))
```

Function: foa.plots

Description:

Produces 10 plots of species frequency of occurrence and abundance, and is designed for community data sets containing samples (rows) by species abundance (columns) data.

Usage:

foa.plots(x, var=' ', margin='column', outfile=' ', na.rm=TRUE, ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing samples (rows) and species abundance (column) data.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

margin: *optional* choice of column or row summary. Note, this does not effect the plots produced; it only effects the summary table returned. [default = 'column']

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

na.rm: *optional* logical (TRUE or FALSE) whether to ignore missing values. [default=TRUE]

... *optional* additional arguments to be passed to the hist() and plot() functions, including the following defaults:

type='o'	Line type
col.point='blue'	color of points and lines in high-level plot
col.line='red'	color of lines in line overlays
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

foa.plots produces the following plots of species and occurrence and abundance, in addition to the table of summary statistics returned to the object (see Value):

Plot1: Empirical distribution of species occurrence – plot of species frequency of occurrence (y-axis)(i.e., number of plots where present) against the cumulative number of species (x-axis). X-axis is the rank-order of species based on their frequency of occurrence among plots. The 5^{th}, 50^{th} and 95^{th} percentiles of the

distribution are shown as horizontal line overlays.

Plot2: Empirical distribution of species relative occurrence – same as plot 1 except that the y-axis is the species *relative* frequency of occurrence (i.e., percentage occurrence in the sample plots). In addition, the 5%, 50% and 95% thresholds are shown as horizontal line overlays.

Plot3: Histogram of species occurrence – histogram of species occurrence (i.e., number of plots where present) with a kernel density overlay. The x-axis represents species frequency of occurrence; y-axis represents density.

Plot4: Histogram of log-transformed species occurrence – same as plot3 except the species occurrence data is log(natural)-transformed.

Plot5: Empirical distribution of species mean abundance – plot of species mean abundance where it occurs (i.e., mean of non-zero plots)(y-axis) against the cumulative number of species (x-axis). X-axis is the rank-order of species based on their mean abundance among plots where they occur. The 5^{th}, 50^{th} and 95^{th} percentiles of the distribution are shown as horizontal line overlays.

Plot6: Frequency of occurrence versus mean abundance – plot of species mean abundance where it occurs (i.e., mean of non-zero plots)(y-axis) against species frequency of occurrence (x-axis).

Plot7: Frequency of occurrence versus log of mean abundance – same as plot6 except the species mean abundance is log(natural)-transformed.

Plot8: Empirical distribution of species per plot – plot of the number of species per plot (i.e., plot richness)(y-axis) against the cumulative number of plots (x-axis). X-axis is the rank-order of plots based on their richness (i.e. number species present). The 5^{th}, 50^{th} and 95^{th} percentiles of the distribution are shown as horizontal line overlays.

Plot9: Empirical distribution of total plot abundance – plot of the total abundance of all species per plot (y-axis) against the cumulative number of plots (x-axis). The x-axis is the rank-order of plots based on their total abundance. The 5^{th}, 50^{th} and 95^{th} percentiles of the distribution are shown as horizontal line overlays.

Plot10: Plot richness versus plot total abundance – plot of the total abundance of all species per plot (y-axis) against the number of species per plot (i.e., plot richness)(x-axis).

Value:

A new *data frame* containing the following variables, depending on whether the margin call was for column or row summaries:

For margin='column':

spc.pres: species frequency of occurrence (i.e., number of plot where present)

spc.perc: species percent occurrence (i.e., percentage of plots species present)
spc.log: log(natural) of species frequency of occurrence
spc.mean: species mean abundance where present (i.e, non-zero plots)

For margin='row'
plt.pres: number of species per plot
plt.sum: total abundance per plot

Author:
K. McGarigal, September 23, 2009

References:
None.

Examples:
```
x<-read.csv('testbird.csv', header=TRUE)
foa.plots(x, 'AMGO:WWPE', margin='row')
```

Function: hclus.cophenetic

Description:

Computes and plots the cophenetic correlation of hierarchical clustering. The cophenetic distance between two observations that have been clustered is defined to be the intergroup dissimilarity at which the two observations are first combined into a single cluster. The cophenetic correlation is the correlation between the dissimilarities in the original *p*-dimensional space and the cophenetic distances. It can be argued that a dendrogram is an appropriate summary of some data if the correlation between the original distances and the cophenetic distances is high. Otherwise, it should simply be viewed as the description of the output of the clustering algorithm. Cophenetic correlations >.75 are considered good.

Usage:

hclus.cophenetic(d, hclus, fit='lm', ...)

Dependency:

None if hclust() employed.
Cluster if agnes() or dian() employed.

Arguments:

d: *required* name of dissimilarity/distance matrix (class = 'dist').

hclus: *required* R object representing a hierarchical clustering. For the default method, an object of class 'hclust' or with a method for as.hclust() such as agnes() and diana().

fit: optional selection of method for fitting a line to the cophenetic scatterplot: 'lm', 'rlm', and 'lqs' (see details). [default = 'lm']

... *optional* additional graphical arguments to be passed to the plot() function.

Details:

The cophenetic scatterplot shows the observed dissimilarities on the x-axis against the cophenetic distances on the y-axes. The strength of the linear relationship is given by the cophenetic correlation coefficient, which is simply a bivariate Pearson product-moment correlation. A fitted regression line is shown using one of three optional linear fitting methods:

lm: fits a linear model using conventional least squares (see help(lm)).

rlm: fits a linear model by robust regression using an M estimator (see help(rlm)).

lqs: fits a regression to the _good_ points in the dataset, thereby achieving a regression estimator with a high breakdown point (see help(lqs)).

Value:

Returns a scalor containing the cophenetic correlation coefficient.

Author:
K. McGarigal, October 18, 2006

References:
None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)] #selecting numeric variables of interest
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.eucl.ave<-hclust(y.eucl,method='average')
hclus.cophenetic(y.eucl,y.eucl.ave)
```

Function: hclus.scree

Description:

Computes a scree plot for an *agglomerative* hierarchical clustering (HC). The scree plot depicts the dissimilarity value of the fusion/split against the number of clusters.

Usage:

hclus.scree(x, ...)

Dependency:

None if hclust() employed
Cluster if agnes() employed.

Arguments:

d: *required* R object representing a hierarchical clustering; currently, an object of class 'hclust' or with a method for as.hclust() such as agnes().

... *optional* additional graphical arguments to be passed to the plot() function.

Details:

For an *agglomerative* HC, as the number of clusters increases, the dissimilarity value at which clusters fuse together typically increases monotonically ('reversals' or 'inversions' are possible with some fusion strategies; e.g., centroid and median linkage). In this case, the scree curve is read from right to left. Abrupt changes in the scree slope or a pronounced "elbow" in the scree plot is an indication of where there is a large change in dissimilarity as clusters fuse together. This usually corresponds to a level in the dendrogram where the branches are relatively long. This is a logical level at which to "cut" the dendrogram. Note, the scree plot does not express any information that is not already present in the dendrogram itself, but some find it a useful way to summarize the dendrogram. Unfortunately, I have not yet figured out how to extract the necessary information from the diana() object, so at the current time a scree plot can only be produced from an agglomerative clustering using hclust() or agnes().

Value:

None.

Author:

K. McGarigal, October 18, 2006

References:

None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)] #selecting numeric variables of interest
```

```
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.eucl.ave<-hclust(y.eucl,method='average')
hclus.scree(y.eucl.ave)
```

Function: hclus.table

Description:

Produces a table depicting the agglomeration sequence for an *agglomerative* hierarchical clustering (HC) using the hclust() function. This function simply extracts agglomeration sequence and dissimilarity information from the hclust object and displays in a convenient table format.

Usage:

hclus.table(x)

Dependency:

None.

Arguments:

x: *required* R object representing an agglomerative hierarchical clustering; currently, required object of class 'hclust'.

Details:

For an *agglomerative* HC, as the number of clusters increases, the dissimilarity value at which clusters fuse together typically increases monotonically ('reversals' or 'inversions' are possible with some fusion strategies; e.g., centroid and median linkage). The agglomeration table gives the sequence of fusions and the corresponding distance at which the fusion took place. The information in this table is used to create the dendrogram plot.

The agglomeration sequence reported is interpreted as follows. If a number *j* in the row is negative, then the single observation $|j|$ is merged at this stage. If *j* is positive, then the merger is with the cluster formed at stage *j* of the algorithm.

Value:

A list containing three components:

dist.method: the dissimilarity/distance measure used.

method: the fusion method (e.g., 'average', 'ward', etc.) used to derive the hierarchical clustering.

cluster.table: the agglomeration table, containing four columns. The first column gives the number of clusters at that level of the agglomeration sequence; the second and third columns give the identity of the two entities or clusters being fused (see 'details' for interpretation); and the fourth column gives the dissimilarity value at which the fusion took place.

Author:

K. McGarigal, October 18, 2006

References:

None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)] #selecting numeric variables of interest
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.eucl.ave<-hclust(y.eucl,method='average')
hclus.table(y.eucl.ave)
```

Function: hist.plots

Description:

Produces histograms for individual variables (columns) of a data frame with a kernel density overlay. If a grouping variable is specified, histograms are constructed separately for each group and plotted together on the same page for efficient comparison.

Usage:

hist.plots(x, var=' ', by=' ', save.plot=FALSE, na.rm=TRUE, ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]

save.plot: *optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'hist.var.jpg', where 'var' is variable name. [default = FALSE]

na.rm: *optional* logical (TRUE or FALSE) whether to ignore missing values. [default=TRUE]

... *optional* additional arguments to be passed to the hist() and plot() functions, including the following defaults:

col.hist='blue'	color of histogram bars
col.line='black'	color of line in kernel density overlay
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

hist.plots is simply a convenience wrapper for the hist() and plot() functions that makes it efficient to quickly produce histograms with kernel density overlays for many variables, and optionally by a grouping variable. Note, the axes for the histogram and the kernel density curve do not align perfectly and I have not found a solution in R yet. So, the kernel density curve may appear shifted to one side or slightly compressed or expanded compared to the histogram.

Value:

No object is returned.

Author:
K. McGarigal, September 14, 2006

References:
None.

Examples:
```
x<-read.csv('testbird.csv', header=TRUE)
hist.plots(x, 'AMGO:WWPE', c('BASIN','SUB'), TRUE, col.line='red')
```

Function: intrasetcor

Description:
Finds the so-called "intRA-set correlation" or (weighted) correlation of weighted averages (sums) scores (WA scores) and constraints (LC scores). This function is fashioned (copied) after the intersetcor() function in the vegan library, which computes the so-called "intER-set correlations.

Usage:
intrasetcor(object)

Dependency:
None.

Arguments:
object: *required* object from 'cca', 'rda', 'capscale' or 'decorana'.

Details:
Intra-set correlations are related to the canonical coefficients but differ in several important aspects. Both canonical coefficients and intra-set correlations relate to the rate of change in community composition per unit change in the corresponding environmental variable, but for canonical coefficients it is assumed that other environmental variables are being held constant, while for intra-set correlations other environmental variables are assumed to covary as they do in the data set. When variables are inter-correlated, canonical coefficients become unstable – the multicollinearity problem; however, intra-set correlations are not effected.

Canonical coefficients and intra-set correlations indicate which environmental variables are more influential in structuring the ordination, but cannot be viewed as an independent measure of the strength of relationship between communities and environmental variables. "[intra-set correlations] have all the problems of correlations, like being sensitive to extreme values, and they focus on the relationship between single constraints and single axes instead of multivariate analysis." (Oksanen). Intra-set correlations are probably only meaningful at all if the site scores are derived as LC scores. Fitted vectors ('envfit') provide a better alternative.

Value:
A vector or matrix of correlations where rows represent the constraints (explanatory variables) and the columns represent the canonical axes.

Author:
K. McGarigal, November 28, 2006

References:
None.

Examples:
moths<-read.csv('moths.csv',header=TRUE)

```
hab.full<-read.csv('moths.full.csv',header=TRUE)
y.rda<-rda(moths[,-1]~.,data=hab.full[,-1])
round(intersetcor(y.rda),3)
```

Function: kappa.mc

Description:

Conducts a Monte Carlo randomization test of the Kappa statistic (chance-corrected correct classification rate) and outputs the cutpoint that maximizes Kappa, max Kappa, and the confusion matrix, omission error rate and commission error rate at the optimal cutpoint. This function works on a fitted generalized linear model (using the glm function) when the dependent variable is binary; i.e., for a binary logistic regression model.

Usage:

kappa.mc(fit, test=FALSE, reps=1000, plot=TRUE, ...)

Dependency:

none

Arguments:

fit: *required* fitted glm object from binary logistic regression; i.e., binary dependent variable and binomial error family.

test: *optional* logical (TRUE or FALSE) indicating whether to conduct a Monte Carlo randomization test.. [default=FALSE]

reps: *optional* integer giving the number of Monte Carlo permutations if test=TRUE. [default=1000]

plot: *optional* logical (TRUE or FALSE) indicating whether to produce of histogram of the random permutation (i.e., null) distribution of Kappa along with the original observed Kappa. [default = TRUE]

... *optional* additional arguments to be passed to the hist() function.

Details:

kappa.mc() is a function for conducting a Monte Carlo randomization test of the Kappa statistic. Specifically, given a glm fitted object from a binary logistic regression, this function extracts the data, the model formula, and any offset and/or weights used in the original fit from the fitted object., computes the maximum Kappa (i.e., finds the cutpoint that maximizes Kappa), and then repeatedly permutes the dependent variable (i.e., random shuffle) and recomputes Kappa. The p-value is computed as the proportion of the permutation distribution (i.e, null distribution) that is greater than or equal to the observed Kappa (i.e., upper one-sided test). A histogram of the null distribution of Kappa and a vertical line showing the observed Kappa is optionally produced.

Value:

kappa.mc returns a list containing the following components (depending on whether test=TRUE):

cutpoint: The cutpoint (real number) for classifying observations as 'present' or 'absent'; any observation with a predicted value greater than or equal to the cutpoint is predicted to be 'present', while less than the cutpoint is predicted to be 'absent'. The cutpoint is given on scale of the fitted values, obtained by transforming the linear predictors by the inverse of the link function.

Kappa: maximum Kappa for the fitted model, which is found at the cutpoint above.

P.value: Proportion of the permuted Kappa (representing the null distribution) greater than or equal to the observed Kappa. [if test=TRUE]

confusion.matrix: the confusion matrix at the cutpoint above, given the number of observed present and absent (columns) and the number of predicted present and absent (rows)

omission: Omission error rate, which equals the number of observed present predicted to be absent (i.e., false negatives) divided by the total number of observed present.

commission: Commission error rate, which equals the number of observed absent predicted to be present (i.e., false positives) divided by the total number of observed absent.

Author:
K. McGarigal, April 2, 2014

References:
None.

Examples:

```
z<-read.csv('eWothClim.csv', header=TRUE)
fit<-glm(pres~day+ I(day^2) + time + I(time^2) + gdd + I(gdd^2) + pmaysept +
    I(pmaysept^2) + tminjan + tavgjan + I(tavgjul^2), data=z, family=binomial(link=logit),
    offset=effortHrs, weights=w25)
kappa.mc(fit1,test=TRUE,reps=1000,plot=TRUE)
```

Function: lda.structure

Description:

Structure coefficients (also referred to as "structure correlations" or "correlation loadings" or just plain "loadings) for the corresponding eigenvectors derived from lda(). Structure coefficients are simple linear correlation coefficients between the original variables and the canonical functions (i.e., the linear discriminants). The squared structure coefficients give the percentage of variance in each original variable accounted for by each principal component. Note that these are linear correlations, which can be misleading if the relationships are non-linear.

Usage:

lda.structure(x.lda, x, dim=ncol(x.lda), digits=3, cutoff=0)

Dependency:

library(MASS).

Arguments:

x.lda: *required* object of class lda, i.e., the result of lda() applied to data frame containing one or more numeric variables and a grouping vector.

x: *required* name of data frame containing one or more numeric variables; actually, it must be the same set of variables used to derive the lda.

dim: *optional* number of dimensions to print. [default = all *Q* dimensions, equal to *G*-1 or *P*, whichever is smaller; where *G* is the number of groups and *P* is the number of discriminating variables]

digits: *optional* number of significant digits to report for the structure coefficients. [default = 3]

cutoff: *optional* minimum structure coefficient for suppressing the printing of small coefficients (loadings). [default = 0]

Details:

Structure coefficients can be computed in different ways. They can either be computed directly from the eigenvector coefficients or they can be calculated directly as the Pearson product-moment bivariate correlation between each variable and the canonical scores for each canonical function. Lda.structure() adopts the latter approach.

In contrast to the eigenvector coefficients (i.e., the raw canonical coefficients), the structure coefficients are correlations and directly measure the strength of the linear relationship between each variable and each canonical function. Large positive values indicate that samples positioned on the positive end of the canonical axis contain larger values of the corresponding variable, and vice versa for negative values. Structure coefficients are usually used as the basis for interpreting the meaning of the canonical functions.

Value:
A data frame containing the structure coefficients.

Author:
K. McGarigal, November 4, 2006

References:
None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
y.lda<-lda(y,grouping=grp)
y.lda.pred<-predict(y.lda)
scores<-y.lda.pred$x
lda.structure(scores,y)
```

Function: mantel2

Description:

This function is simply a convenience wrapper for the mantel() function in the vegan library to supplement the information in the output object in order to facilitate plotting the results with plot.mantel(). Mantel() finds the Mantel statistic as a matrix correlation between two dissimilarity matrices. The significance of the statistic is evaluated by permuting rows and columns of the first dissimilarity matrix.

Usage:

mantel2(xdis, ydis, method='pearson', permutations=1000, strata=NULL)

Dependency:

library(vegan).

Arguments:

Exactly as in mrpp():

xdis, ydis: *required* dissimilarity matrices or 'dist' objects (i.e., class = 'dist').

method: *optional* correlation method, as accepted by cor(): 'pearson', 'spearman' or 'kendall'. [default = 'pearson']

permutations: *optional* number of permutations in assessing significance. [default = 1000]

strata: *optional* integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata. [default = null]

Details:

mantel2() simply adds the xdis and ydis objects as components to the list object that is output from the mantel() function for the purpose of facilitating the plots constructed using plot.mantel() in BIOSTATS.

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 of the Mantel test statistic.

permutations: number of permutations.

xdis: the dissimilarity matrix input as the X-matrix.

ydis: the dissimilarity matrix input as the Y-matrix.

Author:
K. McGarigal, October 25, 2006

References:
see mantel() in vegan library.

Examples:

```
turtle<-read.csv('byturtle.csv', header=TRUE)
y<-turtle[,6:30]
grp<-turtle[,3]
y.std<-data.stand(y, method='standardize', margin='column', plot=FALSE)
y.eucl<-data.dist(y.std, method='euclidean')
contrast<-contrast.matrix(grp)
y.mantel<-mantel2(contrast, y.eucl)
```

Function: mantel.part

Description:

This function computes a series of Mantel and partial Mantel tests between a single y-set (typically species data set) and two or more explanatory variables sets, and optionally including another conditional data set that is partialled out of all tests. This function is essentially a convenience wrapper for the mantel() function in the ecodist library.

Usage:

mantel.part(y, x1, x2, x3=' ', p=' ', digits=3, ydist='euclidean', xdist='euclidean', pdist='euclidean', yscale=FALSE, xscale=TRUE, pscale=FALSE, ...)

Dependency:

library(ecodist).

Arguments:

y: *required* y-set of variables (data frame or matrix) or dissimilarity matrix (i.e., class = 'dist').

x1-x2: *required* x-sets of explanatory variables (data frame or matrix) or dissimilarity matrices (i.e., class = 'dist'). The data frame must have the same number of rows (samples) as the y-set or have the same dimensions as the y-distance matrix.

x3: *optional* x-set of explanatory variables, same as above.

p: *optional* set of conditional variables (data frame or matrix) or dissimilarity matrix (i.e., class = 'dist'). If provided, this data set is partialled out of all analyses (see details).

digits: *optional* number of digits to report in the output. [default = 3]

ydist: *optional* distance/dissimilarity measure to use for the y-set of variables (see data.dist() for the available options); ignored if 'y' is a distance matrix (class = 'dist'). [default = 'euclidean']

xdist: *optional* distance/dissimilarity measure to use for the x-set of variables (see data.dist() for the available options); ignored if 'x' is a distance matrix (class = 'dist'). The same distance measure is used for all explanatory data sets. [default = 'euclidean']

pdist: *optional* distance/dissimilarity measure to use for the p-set of conditional variables if provided (see data.dist() for the available options); ignored if 'p' is a distance matrix (class = 'dist'). [default = 'euclidean']

yscale: *optional* logical (TRUE or FALSE) indicating whether the y-set variables should be column standardized (i.e., z-score standardization) before computing the distance matrix; ignored if 'y' is a distance matrix (class = 'dist'). [default = FALSE]

xscale: *optional* logical (TRUE or FALSE) indicating whether the x-sets of explanatory variables should be column standardized (i.e., z-score standardization) before computing the distance matrix; ignored if 'x' is a distance matrix (class = 'dist'). [default = TRUE]

pscale: *optional* logical (TRUE or FALSE) indicating whether the p-set of conditional variables, if provided, should be column standardized (i.e., z-score standardization) before computing the distance matrix; ignored if 'p' is a distance matrix (class = 'dist'). [default = FALSE]

... *optional* additional arguments to be passed to the mantel() function in the ecodist library. In particular, there are options for ranked data, permutation tests, and bootstrapped confidence limits.

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 and the observations are clearly not independent. Consequently, a Monte carlo permutation test is conducted in which the first dissimilarity matrix is permuted. Partial Mantel statistic uses partial correlation conditioned on one or more additional matrices. Only the first matrix is permuted so that the correlation structure between second and first matrices is kept constant.

The mantel.part() function simply calls the mantel() function in ecodist many times, each time designating a different model. The models analyzed include the *marginal effects* of each of the explanatory data sets; that is, the correlation between each explanatory matrix and the y-matrix without partialling out any of the other explanatory sets. However, if a conditional data set is provided, it is partialled out. These so-called marginal tests measure the correlation between two matrices without considering confounding between explanatory data sets. Thus, the marginal tests provide a measure of the total correlation between an explanatory data set and the y-set.

In addition, the models analyzed include a series of conditional effects in which one or more explanatory data sets are partialled out before computing the correlation between the explanatory data set and the y-matrix. As before, if a conditional data set is provided, it is partialled out as well. These so-called conditional tests measure the partial correlation between two matrices after considering the correlation due to other conditional data sets. Thus, the conditional tests provide a measure of the independent or unique correlation between an explanatory data set and the y-set.

Note, although mantel() function (in ecodist library) called by mantel.part() silently accepts other methods than 'pearson' (see mrank argument in the mantel() function in ecodist which can be passed from mantel.part() to compute rank correlations), partial correlations will probably be wrong with other methods according to Oksanen.

Value:

The function returns a list of class 'mantel.part' with following components:

call: function call.

ptable: name of conditional data set if provided.

xtables: names of the explanatory data sets provided.

marginal: data frame containing the Mantel statistics for the marginal effects. Each marginal test is a test of the relationship between a single explanatory data set and the species data set. If a conditional data set is provided, the test is a partial Mantel test that removes the effect of the conditional data set before computing the congruence between the explanatory and species data sets. The following results (columns) are reported for each marginal test (rows):

- Standardized Mantel *r* statistic: This is a correlation coefficient and is a measure of the strength of the relationship between distance matrices.
- pval1-pval3: These are p-values derived from a Monte carlo permutation test, where each p-value represents a different null hypothesis test.
  - pval1: one-tailed p-value (null hypothesis: $r \leq 0$).
  - pval2: one-tailed p-value (null hypothesis: $r \geq 0$).
  - pval3: two-tailed p-value (null hypothesis: $r = 0$).

  In most applications we are interested in the first test. That is, we expect the correlation between the explanatory distance matrix and the species distance matrix to be positive. Thus, the appropriate null hypothesis is that the Mantel r is ≤ 0.
- llim-ulim: lower and upper confidence limits on the Mantel r statistic. By default this is a 95% confidence interval.

partial: data frame containing the partial Mantel statistics for the conditional effects. Each partial Mantel test is a test of the relationship between a single explanatory data set and the y-set after conditioning on one or more other data sets; that is, after partialling out the effects of one or both of the remaining explanatory data sets. If a conditional data set is provided, it is included with the other data sets being partialled out of each test. Otherwise the results are as described above for the marginal Mantel tests.

Author:

K. McGarigal, December 5, 2006

References:

see mantel() in ecodist library.

Examples:

```
moths<-read.csv('moths.csv',header=TRUE)
hab.plot<-read.csv('moths.plot.csv',header=TRUE)
hab.patch<-read.csv('moths.patch.csv',header=TRUE)
hab.land<-read.csv('moths.land.csv',header=TRUE)
hab.space<-read.csv('moths.space.csv',header=TRUE)
```

```
y<-moths[,-1]
x.plot<-hab.plot[,c(2,9,10)]
x.patch<-hab.patch[,c(4,6,7)]
x.land<-hab.land[,c(3,8,9)]
x.space<-hab.space[,c(3,7,8)]
y.log<-data.trans(y,method='log',plot=FALSE)
y.chord<-data.stand(y.log,method='normalize',margin='row',plot=FALSE)
z<-mantel.part(y.chord,x.plot,x.patch,x.land,p=x.space,ydist='euclidean',xdist='euclidean',
pdist='euclidean',yscale=FALSE,xscale=TRUE,pscale=FALSE)
```

Function: mrpp2

Description:

This function is simply a convenience wrapper for the mrpp() function in the vegan library to supplement the information in the output object in order to facilitate plotting the results with plot.mrpp(). Mrpp() provides a test of whether there is a significant difference between two or more groups of sampling units. The significance of the statistic is evaluated by permuting the sample observations and their associated distances.

Usage:

mrpp2<-function(dat, grouping, permutations=1000, distance='euclidean', weight.type=1, strata)

Dependency:

Vegan.

Arguments:

Exactly as in mrpp():

dat: *required* data matrix or data frame in which rows are samples and columns are response variable(s), or a dissimilarity object (i.e., class = 'dist') or a symmetric square matrix of dissimilarities.

grouping: *required* vector (factor or numeric index) for grouping observations.

permutations: *optional* number of permutations to assess the significance of the MRPP statistic, delta. [default = 1000]

distance: *optional* choice of distance metric that measures the dissimilarity between two observations. See vegdist() in the vegan library or data.dist() in BIOSTATS for options. This will be used only if 'dat' was not a dissimilarity structure of a symmetric square matrix. [default = 'euclidean']

weight.type: *optional* choice of group weights. See details in mrpp() for options. [default = 1]

strata: *optional* integer vector or factor specifying the strata for permutation. If supplied, observations are permuted only within the specified strata. [default = none]

Details:

See the details in the help file for mrpp() for a complete description of this function. mrpp2() simply adds two components (class.vec and diss.vec) to the list object (see below) that is output from the mrpp() function for the purpose of facilitating the plots constructed using plot.mrpp() in BIOSTATS.

Value:

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

call: function call.

delta: the overall weighted mean of group mean distances.

E.delta: expected delta, under the null hypothesis of no group structure. This is the mean of the permuted deltas.

Pvalue: significance of the test.

A: a chance-corrected estimated 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 data sets.

permutations: the number of permutations used.

class.vec: the class membership vector equal in length to the number of elements in the lower triangle of the distance matrix, giving the value 'between' for all elements represent between-group distances and the value of the corresponding group (from the grouping vector) for all other elements.

diss.vec: the dissimilarity vector equal in length to the number of elements in the lower triangle of the distance matrix; simply the dissimilarities given in the distance matrix but in their vector form.

Author:

K. McGarigal, October 25, 2006

References:

see mrpp() in vegan library.

Examples:

```
turtle<-read.csv('byturtle.csv', header=TRUE)
y<-turtle[,6:30]
grp<-turtle[,3]
y.std<-data.stand(y, method='standardize', margin='column', plot=FALSE)
y.eucl<-data.dist(y.std, method='euclidean')
y.mrpp<-mrpp2(y.eucl, grp)
```

Function: mv.outliers

Description:

Screens numeric data in a data frame for potential *multivariate* "outliers." Specifically, mv.outliers computes the distance between samples (row) based on a specified distance measure (from data.dist) and then computes for each sample its mean distance to all other samples. The resulting mean distance scores are standardized to zero mean and unit variance (i.e., z-score standardization) and the results are displayed in paired histograms depicting the distribution of mean distance scores and the distribution of z-scores. In addition, a list of the samples exceeding a specified threshold in standard deviations from the mean average distance is produced. An alternative method is also provided based on Mahalanobis distance. Briefly, the Mahalanobis distance of each sample point to the remaining set of observations is computed and compared to the expected distribution of values based on the chi-square distribution (see details below).

Usage:

mv.outliers(x, method, var=' ', cor.method='pearson', outfile=' ', sd.limit=3, alpha=.001, plot=TRUE, save.plot=FALSE, use='complete.obs', na.rm=TRUE, ...)

Dependency:

library(vegan)
library(MASS)

Arguments:

x: *required* name of data frame containing one or more numeric variables.

method: *required* distance measure: 'mahalanobis', 'euclidean', 'manhattan', 'correlation', 'canberra', 'bray', 'kulczynski', 'jaccard', 'gower', 'morisita', 'horn', 'mountford', 'raup', 'binomial' (see below for details).

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

cor.method: *optional* type of correlation coefficient for method = 'correlation', including: 'pearson', 'kendall' and 'spearman'. [default = 'pearson']

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

sd.limit *optional* threshold level in standard deviations for flagging extreme values. [default = 3]

alpha: *optional* alpha level for determining the critical chi-square value above which samples are deemed "outliers". [default = .001]

plot: *optional* logical (TRUE or FALSE) to determine whether paired histograms depicting the distribution of average distances and standard deviations in average distances are plotted or not, or if method = 'mahalanobis', a kernel density plot and quantile-quantile plot. [default = TRUE]

save.plot: *optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'dhist.method.jpg', where 'method' is distance method chosen. [default = FALSE]

use: *optional* character string giving a method for computing covariances in the presence of missing values if method = 'correlation'. This must be (an abbreviation of) one of the strings: 'all.obs', 'complete.obs' or 'pairwise.complete.obs'. [default = 'complete.obs']

na.rm: *optional* logical (TRUE or FALSE) whether to ignore missing values. [default=TRUE]

... *optional* additional arguments to be passed to the decostand(), hist(), and plot() functions, including the following graphical defaults:

col.hist='blue'	color of histogram
col.line='black'	color of the kernel density line
col.point='blue'	color of the points in the mahalanobis plots
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

Extreme values can exert undue influence on the results of most multivariate techniques. Thus, it is good practice to screen the data for extreme values before conducting any analysis. There are many ways to identify extreme observations. Mv.outliers identifies samples (rows) that have extreme values based on their average multivariate distance to all other samples or their mahalanobis distance to the group of all other samples. Consequently, this is a true "multivariate" method of identifying extreme values. Importantly, just because an observation is extreme, doesn't mean that it should automatically be deemed an "outlier" and deleted from the data set. In general, no observation should be dropped unless it can be justified on ecological grounds, e.g., that it represents a real ecological oddity.

If you are planning on using a distance matrix is in a subsequent analysis (e.g., principal coordinates analysis, non-metric multidimensional scaling, Mantel test, cluster analysis, etc.), then the chosen distance method for multivariate outlier detection should be the same.

The mahalanobis method computes the distance between each observation and the data cloud, and does so using a robust estimator of the covariance matrix based on the cov.rob() in the MASS library. See help(cov.rob) for details on this function. Briefly, it involves computing a multivariate location (center of the data cloud) and scale (variance-covariance) estimate with a high breakdown point - this can be thought of as estimating the mean and covariance of the 'good' part of the data (i.e., without the extreme observations). Unfortunately, there are least two ways this method can fail with typical ecological data sets. First, if the data matrix contains any

variable with an interquartile range of zero (i.e., 1st and 3rd quartiles of the data the same), the cov.rob() function will fail. Unfortunately, this condition is quite common in community data sets because often many species occur infrequently among samples (i.e., lots of zero counts). An unsatisfying solution is to drop all offending variables, but this can result in dropping a significant number of variables (species) and may involve dropping variables that are the most likely to produce extreme (or outlying) observations. A second unsatisfying solution is to compute the covariance matrix using the standard cov() function, which is unsatisfying because the full set of observations, including the potential outliers, is included in the covariance estimation. Second, Mahalanobis distance requires the covariance matrix to be invertable (i.e., non-singular). A matrix is singular and therefore "ill-conditioned" if the determinant is zero (i.e., there is a zero eigenvalue). This can occur whenever there is a perfect linear dependency among two or more variables. The simple solution of course, in this case, is to drop one or more of the offending variables and try again. A more elegant solution (not implemented here) is to compute the principal components, drop the components with eigenvalues less than some small threshold value, and then compute Mahalanobis distance on the reduced set of principal component scores.

If method = 'mahalanobis' is chosen the Mahalanobis distance can be calculated and a kernel density plot and quantile-quantile plot are generated. The density plot displays the distribution of Mahalanobis distances with the critical threshold distance based on the specified alpha-level depicted as a vertical line. Note, the critical value is based on the chi-square distribution which assumes an underlying multivariate normal distribution. In addition, a quantile-quantile (QQ) plot is also generated, which plots the sample quantiles of the observed Mahalanobis distribution against the expected quantiles of a theoretical distribution derived from the chi-square distribution. If the actual data are perfectly multivariate normal, the QQ plot should be a diagonal straight line (i.e., slope of one). Departure from this diagonal indicates deviations from expected and individual points that fall far off this line may be deemed extreme points.

Value:

Returns a new *data frame* containing the reduced set of samples (rows) with extreme values, either based on standard deviations in average distance to all other samples or the Mahalanobis distance.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
mv.outliers(x, 'correlation', 'AMGO:WWPE', 'pearson', 'D/stats/testout', 2.5, .005)
```

Function: nhclus.scree

Description:

Computes a scree plot for a nonhierarchical clustering (NHC) using the pam() function in the Cluster library. The scree plot depicts the sum of within-cluster dissimilarities to cluster medoids for a given number of clusters, which is the objective function minimized by the algorithm. In addition, the scree plot depicts the average silhouette width of all sampling entities for a given number of clusters (see details).

Usage:

nhclus.scree(x, ...)

Dependency:

Cluster.

Arguments:

x: *required* data matrix or data frame, or dissimilarity matrix or object. In the case of a matrix or data frame, each row corresponds to an observation, and each column corresponds to a variable. All variables must be numeric. Missing values (NA's) are allowed-as long as every pair of observations has at least one case not missing. If 'x' is a data matrix or data frame, then at least one additional argument must be passed to the pam() function designating the dissimilarity metric to use (e.g., metric = 'euclidean'), although only 'euclidean' and 'manhattan' are allowed (see pam and clara functions).

In case of a dissimilarity matrix (class = 'dist'), 'x' is typically the output of data.dist(), vegdist(), dist() or daisy(). Also a vector of length n*(n-1)/2 is allowed (where n is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values (NAs) are not allowed in this case.

... *optional* additional arguments to be passed to the pam() function.

Details:

In contrast to hierarchical clustering (HC), with NHC we typically need to specify a priori the number of clusters sought. However, in many cases we don't have a good idea of how many clusters to expect and we would like characteristics of the data to determine how many clusters to keep. For this purpose, we can construct a *scree plot* of an objective criterion against the number of clusters and look for indications of the "best" number of clusters to retain. Once possible criterion is the *average within-cluster dissimilarity*, which declines monotonically as the number of clusters increases; it typically declines rapidly at first and then levels off somewhat. An abrupt leveling off or a so-called "elbow" in the scree plot can provide an indication of the number of clusters to keep.

Another criterion available in the Cluster library is the *average silhouette width*. For each observation *i*, the silhouette width *s(i)* is defined as follows:

Put *a(i)* = average dissimilarity between *i* and all other points of the cluster to which *i*

belongs (if *i* is the only observation in its cluster, *s(i)* = 0 without further calculations). For all other clusters *C*, put *d(i,C)* = average dissimilarity of *i* to all observations of *C*. The smallest of these *d(i,C)* is *b(i) = min_C d(i,C)*, and can be seen as the dissimilarity between *i* and its "neighbor" cluster, i.e., the nearest one to which it does not belong. Finally,

$$s(i) = (b(i) - a(i)) / max(a(i), b(i)).$$

Observations with a large *s(i)* (almost 1) are very well clustered, a small *s(i)* (around 0) means that the observation lies between two clusters, and observations with a negative *s(i)* are probably placed in the wrong cluster. The average silhouette width for all observations can be plotted against the number of clusters in a scree plot. We expect the average silhouette width to be highest when the number of clusters equals the natural clustering of the data.

Note, the scree plot will depict the objective criteria for 2 to max.k number of clusters. Obviously, max.k cannot exceed the number of sample observations *N* and it probably makes no sense for max.k to approach *N*.

Value:

Returns a data frame containing three columns of data used to create the scree plot. The first column gives the number of clusters (ranging from 2 to max.k). The second column gives the sum of the within-cluster dissimilarities for the corresponding number of clusters. The third column gives the average silhouette width across all observations for the corresponding number of clusters.

Author:

K. McGarigal, October 18, 2006

References:

None.

Examples:

```
bird.niche<-read.csv('bird.niche.csv',header=TRUE)
x<-bird.niche[bird.niche$NOBLOCKS>4,]
y<-x[,-c(1,50:56)] #selecting numeric variables of interest
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
nhclus.scree(y.eucl,max.k=20)
```

Function: nmds.monte

Description:

Monte carlo permutation test of significant of the final stress statistic derived from nonmetric multidimensional scaling (NMDS). Stress is a measure of goodness-of-fit between the original sample distances in *p*-dimensional space and distances in the reduced *k*-dimensional ordination space. This function is wrapper for the metaMDS() function in the vegan library.

Usage:

nmds.monte(x, k, distance='bray', autotransform=FALSE, trace=0, zerodist='add',perm=100, col.hist='blue', col.line='red', lty=2, las=1, lab=c(5,5,4), ...)

Dependency:

library(vegan)

Arguments:

x: *required* data frame or matrix containing one or more numeric variables.

k: *required* number of dimensions to test.

distance: *optional* distance measures to use in the NMDS ordination; can use any of the measures available in the vegdist() function in the vegan library. [default = 'bray']

autotransform: *optional* logical (TRUE or FALSE) to specify whether or not to use simple heuristics for possible data transformation (see below). [default = FALSE]

trace: *optional* level of tracing (outputting) the NMDS function. Trace = 0 will suppress the printing of any intermediate results; trace >=2 will produce voluminous output. [default = 0]

zerodist: *optional* handling of zero dissimilarities: either 'fail' or 'add' a small positive value. Option 'fail' will cause the function to fail if zero dissimilarities are encountered, which happens anytime two rows are identical.

perm: *optional* number of permutations of the data matrix for the randomization test. Note, the processing time can be unbelievably long, so the default is set at 100, and even that can take hours on large data sets. [default = 100]

... *optional* additional arguments to be passed to the hist() and metaMDS() functions, including the following defaults:

col.hist='blue'	color of histogram bars
col.line='red'	color of vertical line represent observed eigenvalue
lty=2	line type for the vertical line (dashed)
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

nmds.monte() randomly shuffles each column of the data matrix to remove any real correlation structure and then computes NMDS for the specified number of dimensions, storing the final stress value. This is repeated many times (default = 100) and the resulting distribution of stress values represents the expected distribution under the null hypothesis of no real correlation structure. Note, by shuffling each column independently, we are maintaining the exact data domain (i.e., we are not sampling any new values on any variable). By comparing the observed stress statistic to the random distribution of stress values, we can determining directly the probability of observing our stress value if in fact it was derived from a data set without any real correlation structure (i.e., the p-value). This is simple and direct way to test the statistical significance of the final stress statistic, but tends to be a conservative test by ecological criteria. Consequently, this test is best used in conjunction with other criteria for determining the importance of each ordination axis.

If zero dissimilarities are encountered the function will fail unless the optional argument zerodist='add' is used (which is the default in nmds.monte). This argument will add a very small number to each distance element of the computed distance matrix.

This Monte carlo randomization test is helpful for selecting dimensionality but is not foolproof (McCune and Grace 2002). Strong outliers caused by one or two extremely high values and/or a single dominant species (in community data sets) can result in randomizations with final stress values similar to the real data. In addition, with very small data sets (say <10 samples), this test can be too conservative. Randomizations in this case can produce configurations with final stress equal to zero.

Value:

Prints to the console the permuted stress values and returns a data frame containing the observed stress statistic and the corresponding p-value.

Author:

K. McGarigal, October 4, 2006

References:

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

Examples:

```
nmds.monte(x, dim=3)
```

Function: nmds.scree

Description:

Scree plot of the final stress statistic against the number of dimensions in a Nonmetric Multidimensional Scaling (NMDS) analysis. This function is a convenience wrapper for the metaMDS() function. Specifically, nmds.scree() calls the metaMDS() function several times, once for each number of specified dimensions, and then plots the final stress from each ordination against the number of dimensions. Note, given the computational demands of NMDS, this scree plot can take a long time to generate for large data sets.

Usage:

nmds.scree(x, distance='bray', k=6, trymax=50, autotransform=FALSE, trace=0, ...)

Dependency:

library(vegan).

Arguments:

x: *required* data frame or matrix containing one or more numeric variables).

distance: *optional* distance metric to use in nonmetric multidimensional scaling (NMDS). Any of the distance metrics available in the vegdist() function of the vegan library can be used. [default = 'bray']

k: *optional* maximum number of dimensions to compute NMDS. A separate NMDS will be computed using the metaMDS() function in the vegan library for 1 to k dimensions. [default = 6 dimensions]

trymax: *optional* maximum number of random starts in search of stable solution for NMDS. [default = 50]

autotransform: *optional* logical (TRUE or FALSE) to specify whether or not to use simple heuristics for possible data transformation (see below). [default = FALSE]

trace: *optional* level of tracing (outputting) the NMDS function. Trace = 0 will suppress the printing of any intermediate results; trace >=2 will produce voluminous output. [default = 0]

... *optional* additional arguments passed to the metaMDS() or plot() functions.

Details:

The scree plot for nonmetric multidimensional scaling (NMDS) is a bit different than the familiar scree plot of eigenvalues. NMDS is not an eigenvector technique. Instead, NMDS computes a stress statistic which is a measure of goodness-of-fit between the original distances in *p*-dimensional space and distance in the reduced *k*-dimensional ordination space. The NMDS scree plot is a plot of stress against the number of dimensions and can be used similarly as an

aide to determine the appropriate number of dimensions to use in the final solution.

If zero dissimilarities are encountered the function will fail unless the optional argument zerodist='add' is used. This argument will add a very small number to each distance element of the computed distance matrix.

MetaMDS() has the option for automatic data transformation. Specifically, if the data values are larger than common class scales, the function performs a Wisconsin double standardization using wisconsin() in the vegan library. If the values look very large, the function also performs 'sqrt' transformation. Both of these adjustments are generally found to improve the results. However, the limits are completely arbitrary (at present, data maximum 50 triggers 'sqrt' and >9 triggers 'wisconsin'). If you want to have a full control of the analysis, you should set 'autotransform = FALSE' and make explicit standardization in the command, which is the default implemented in ordi.scree().

Value:
None.

Author:
K. McGarigal, October 4, 2006

References:
None.

Examples:
nmds.scree(rip.bird, distance='gower') #for NMDS scree plot

Function: norm.test

Description:

Simply a convenience wrapper for a set of normality tests in the nortest library, including the Anderson-Darling test, Cramer-von Mises test, Lilliefors (Kolmogorov-Smirnov) test, Pearson chi-square test, and Shapiro-Francia test for normality. This function computes the select test for each variable in the specified data set. If a grouping variable is specified, the test is conducted on the residuals of a one-way analysis of variance (i.e., the group mean-centered values).

Usage:

norm.test(x, groups=' ', var=' ', method='ad', ...)

Dependency:

library(nortest).

Arguments:

x:	*required* numeric vector of data values, or data frame or matrix containing one or more numeric variables.
groups:	*optional* vector (either numeric or character) containing the group membership of each observation in x. If omitted, the normality test is conducted on the entire vector. If present, the normality test is conducted on the residuals of a one-way analysis of variance (i.e., the group mean-centered values).
var:	*optional* list of variables in x to test. If omitted, all variables in x are tested. [default = test all variables]
method:	*optional* choice of test method: 'ad', 'sf', 'cvm', 'lillie', or 'pearson' (see details). [default = 'ad']
...	*optional* additional arguments passed on to the test functions.

Details:

The following normality tests are included:

ad:	Anderson-Darling test for the composite hypothesis of normality. The Anderson-Darling test is the recommended EDF test by Stephens (1986). Compared to the Cramer-von Mises test (as second choice) it gives more weight to the tails of the distribution.
sf:	Shapiro-Francia test is simply the squared correlation between the ordered sample values and the (approximated) expected ordered quantiles from the standard normal distribution, and is known to perform well.
cvm:	Cramer-von Mises test is an EDF omnibus test for the composite hypothesis of

normality.

lillie: Lilliefors (Kolmogorov-Smirnov) test is an EDF omnibus test for the composite hypothesis of normality. The test statistic is the maximal absolute difference between empirical and hypothetical cumulative distribution function. The Lilliefors (Kolomorov-Smirnov) test is the most famous EDF omnibus test for normality. Compared to the Anderson-Darling test and the Cramer-von Mises test it is known to perform worse.

pearson: Pearson chi-square test for the composite hypothesis of normality The Pearson chi-square test is usually not recommended for testing the composite hypothesis of normality due to its inferior power properties compared to other tests.

Value:

Returns a data frame containing three columns. The first column lists the variable name, the second column gives the test statistic (depending on selected test), and the third column gives the p-value for the null hypothesis test. There is a separate row for each variable tested.

Author:

K. McGarigal, November 4, 2006

References:

See ad.test(), sf.test(), cvm.test(), lillie.test() and pearson.test() in the nortest library.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
norm.test(y,grp,method='cvm')
```

Function: ordi.monte

Description:

Monte carlo permutation test of significant of the eigenvalues derived from an eigenvector ordination analysis. Currently computes Monte carlo tests for principal components analysis (PCA) using the prcomp() function, correspondence analysis (CA) using the cca() function in the vegan library, and detrended correspondence analysis (DCA) using the decorana() function in the vegan library. For nonmetric multidimensional scaling (NMDS), which is not an eigenvector technique, see nmds.monte().

Usage:

ordi.monte(x, ord, dim=length(x), perm=1000, center=TRUE, scale=TRUE, digits=3, plot=TRUE, col.hist='blue', col.line='red', lty=2, las=1, lab=c(5,5,4),...)

Dependency:

none.

Arguments:

x: *required* data frame or matrix containing one or more numeric variables.

ord: *required* ordination method: 'pca', 'ca', or 'dca'

dim: *optional* number of dimensions to test. Note, if ord = 'dca' the maximum number of dimensions is set to 4, since this is all that the decorana() function produces. [default = all *P* dimensions, or 4 in the case of DCA]

perm: *optional* number of permutations of the data matrix for the randomization test. [default = 1000]

center: *optional* choice of centering the matrix prior to conducting the principal components analysis in ord = 'pca'. Rarely would you ever not center. [default = TRUE]

scale: *optional* choice of standardizing the data matrix (i.e., column z-score standardization) so that the correlation matrix rather than covariance matrix is decomposed if ord = 'pca'. Almost always want to scale data matrix for PCA. [default = TRUE]

digits: *optional* number of significant digits to report for p-values. [default = 3]

plot: *optional* choice of whether to plot a histogram of the random permutation distribution for each dimension requested. [default = TRUE]

... *optional* additional arguments to be passed to the hist() functions, including the following defaults:
col.hist='blue' color of histogram bars
col.line='red' color of vertical line represent observed eigenvalue

lty=2	line type for the vertical line (dashed)
las=1	horizontal orientation of axis labels
lab=c(5,5,4)	number of x and y tick marks and length of labels

Details:

Ordi.monte randomly shuffles each column of the data matrix to remove any real correlation structure and then computes the eigenvalues for the randomly permuted data matrix. This is repeated many times (default = 1000) and the resulting distribution of eigenvalues represents the expected distribution under the null hypothesis of no real correlation structure. Note, by shuffling each column independently, we are maintaining the exact data domain (i.e., we are not sampling any new values on any variable). By comparing the observed eigenvalue to the random distribution of eigenvalues, we can determining directly the probability of observing our eigenvalue if in fact it was derived from a data set without any real correlation structure (i.e., the p-value). This is simple and direct way to test the statistical significance of each eigenvalue, but tends to be a conservative test by ecological criteria. Consequently, this test is best used in conjunction with other criteria for determining the importance of each ordination axis.

Value:

Returns a data frame containing the eigenvalues and the corresponding p-values.

Author:

K. McGarigal, October 4, 2006

References:

None.

Examples:

```
x<-prcomp(rip.bird, scale=TRUE)
ordi.monte(x, dim=5)
```

Function: ordi.overlay

Description:

Ordination plot overlay of variables. The overlay variables can be either intrinsic (i.e., same variables used to compute the ordination) or extrinsic (i.e., other variables not used in the ordination). Ordi.overlay() generates two types of plots: (1) a typical ordination plot of sample scores in which the point symbol is scaled proportional to the magnitude of the overlay variable, and (2) a scatterplot of the ordination scores against the overlay variable with a superimposed constrained quantile curve using quadratic (or linear) splines (see details).

Usage:

ordi.overlay(x.ord, x, var=' ', fit=TRUE, choices=c(1,2), expand=5, tau=.95, pch=19, ...)

Dependency:

vegan; fields.

Arguments:

x.ord: *required* ordination object from which scores can be extracted using the scores() function in the vegan library, e.g., the result of prcomp(), rda(), cca(), decorana(), or metaMDS() applied to data frame containing one or more numeric variables.

x: *required* data frame or matrix containing the numeric overlay variables. Note, the overlay variables must be numeric. The data frame or matrix must have the same samples in the same order as the data from which the ordination object was derived.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

fit: *optional* choice of whether to plot scatterplots with fitted envelopes along with the ordination plot of samples scaled by the overlay variable. [default = TRUE]

choices: *optional* choice of ordination axes to plot. [default = c(1,2)]

expand: *optional* choice of scaling constant to control the size of the symbols in the ordination plot. The symbols will be scaled as follows: expand*y[,i]/max(y[,I]). Consequently, a larger scaling constant will increase the size of all sample points. [default = 5]

tau: *optional* choice of quantile for the scatterplot regression spline. A tau of .95 will fit a quadratic (by default) spline to the 95th quantile of the distribution (i.e., essentially an envelop that captures 95% of the values of the overlay variable. [default = .95]

... *optional* additional arguments to be passed to the cobs() and plot() functions, including the following defaults:

pch point symbol type

Details:

The ordi.overlay() function with fit=TRUE displays two scatterplots for the overlay variable corresponding to the chosen ordination axes. In addition, a constrained quantile curve using a quadratic (or linear) regression B-spline (using the cobs() function in the cobs library) is superimposed on each scatterplot. The quantile spline is an estimate of the 95th (dy default) quantile of the distribution. The fitted quantile curve can be quite revealing of nonlinear relationships. For example, unimodal species response functions along underlying gradients will reveal themselves in the fitted curve.

Value:

None.

Author:

K. McGarigal, September 26, 2006;
Updated to the use of the cobs library for the regression spline, March 27, 2013

References:

None.

Examples:

```
x<-prcomp(rip.bird, scale=TRUE)
ordi.overlay(x, rip.hab)
```

Function: ordi.part

Description:

This function computes a variance decomposition or partitioning of a constrained ordination using the rda() or cca() functions in the vegan library. Specifically, given a y-set of variables (typically species data) and two or three explanatory sets of variables, and optionally an additional conditional data set, this function partitions the total variance in the y-set explainable by each explanatory set independent and jointly.

Usage:

ordi.part(y, x1, x2, x3=' ', p=' ', method='rda', model='reduced', digits=3, ...)

Dependency:

library(vegan).

Arguments:

y: *required* y-set of variables (data frame or matrix) composed of numeric variables (typically species data).

x1-x2: *required* x-sets of explanatory variables (data frame or matrix) composed of numeric and/or character variables (typically environmental variables). The data frame must have the same number of rows (samples) as the y-set and be similarly ordered.

x3: *optional* x-set of explanatory variables, same as above. [default = not provided]

p: *optional* set of conditional variables (data frame or matrix) composed of numeric and/or character variables. If provided, this data set is partialled out of all analyses (see details). [default = not provided]

method: *optional* constrained ordination method: 'rda' or 'cca'. Currently, this function only accepts these two options and calls the corresponding functions in the vegan library. [default = 'rda']

model: optional permutation model for the Monte carlo testing: 'direct', 'reduced', or 'full' (see details). [default = 'reduced']

digits: *optional* number of digits to report in the output. [default = 3]

... *optional* additional arguments to be passed to the rda(), cca() and anova.cca() functions in the vegan library. In particular, there are options for controls on the permutation tests.

Details:

The variance partitioning method described here is correlational, but differs from traditional correlation studies in that it explicitly measures both the independent explanatory power and

confounding among several sets of explanatory variables. The variance decompositions that result provide a comprehensive picture of the relative importance, independent effects, and confounding of the factors included in the analysis. The canonical partitioning method was originally used to partition variation in community data sets among environmental and spatial components (Borcard et al. 1992, Borcard and Legendre 1994, Legendre and Borcard 1994). Liu and Brakenhielm (1995) used a related method to partition variance in a plant community into components explainable by geographical position, climate and pollutant deposition. Anderson and Gribble (1998) extended the technique to include the effects of temporal variation, so that variation in community data is partitioned into the components explainable by environmental, spatial, and temporal factors, and their overlap. More recently, Cushman and McGarigal (2002 and 2004) extended the method to address the specific challenges of hierarchically structured data.

The ordi.part() function simply calls the corresponding constrained ordination function in vegan many times, each time designating a different model. Details of the specific models required to partition two or three explanatory data sets are provided elsewhere (see Cushman and McGarigal 2002).

Value:

The function returns a list of class 'ordi.part' with following components:

call: function call.

ptable: name of conditional data set if provided.

xtables: names of the explanatory data sets provided.

total: data frame containing a summary of the overall "unpartitioned" constrained ordination, including the following:

- Total inertia (either 'variance' in RDA/CAP or the 'mean squared contingency coefficient" in CCA) is interpreted as the total "variance" in the species data set. This is the total "variance" in the species data that is partitionable into various components, both explained and unexplained. Reported as the total inertia and as the proportion of the total inertia (always = 1).
- Conditional inertia is the total "variance" in the species data set attributable to the conditional data set, if provided. This measures how much of the species variance is effectively removed before the subsequent partitioning into explained and unexplained components. Reported as the conditional inertia and as the proportion of the total inertia.
- Constrained inertia is the total "variance" in the species data set attributable to all of the explanatory variables or constraints; it is equal to the sum of the canonical eigenvalues. This is the so-called "explained variance". Reported as the constrained inertia and as the proportion of the total inertia.
- Unconstrained inertia is the total "variance" in the species data set left unexplained or unaccounted for by the conditional and/or explanatory variables; it is equal to the sum of the unconstrained eigenvalues. This is the so-called

"residual variance". Reported as the unconstrained inertia and as the proportion of the total inertia.

marginal: data frame containing the results for the *marginal effects*. Marginal effects represent the total "variance" in the species data attributable to a single explanatory data set or a combination of explanatory data sets, without partialling out any other explanatory data sets. This is the variance that can be attributed to a set of explanatory variables without consideration of potential confounding with other variables. For example, the marginal effect of the x1 set of variables is the sum of the canonical eigenvalues based on the set of constraints in x1. This is the variance in the y data that can be accounted for by the x1 variables, but we cannot say whether it is due uniquely or independently to x1 because some (or even all) of the variance explained may be confounded (and therefore inseparable) with other explanatory variables not in the model. Similarly, the marginal effect x12 is the sum of the canonical eigenvalues based on the set of constraints in x1 and x2 combined (as if they were a single set of variables). The marginal effects (rows) are reported as (columns):

- Total inertia attributed to the corresponding marginal variables (equal to the sum of the canonical eigenvalues). This is the marginal effect size.
- Proportion of the total species variance (equal to the sum of the canonical eigenvalues divided by the total inertia). This is the relative effect size given in terms of the total species variance.
- Proportion of the total constrained inertia (equal to the sum of the canonical eigenvalues divided by the total constrained inertia). This is the relative effect size given in terms of the total "explained" variance. This effect size can be quite large even though the percent of species variance explained is quite low. This is a useful measure if the question centers on how much of the explained variance is due to each factor.
- Monte carlo test of significance of the marginal effect size. This is a permutation test of the null hypothesis that the species are unrelated to the explanatory variables (with or without controlling for conditional variables). There are several optional arguments for controlling the permutation test (see help(anova.cca) for more details).

components: data frame containing the results for each partition *component*. Each component corresponds to a single exclusive (non-overlapping) partition of the total species variance, as can be represented in a Venn diagram (see plot.ordi.part).

A two-way partitioning (involving two explanatory data sets) has three components:

- x1 - pure x1 effects; the species variance explained due to x1 constraints alone, independent of the variance due to x2.
- x2 - pure x2 effects; the species variance explained due to x2 constraints alone, independent of the variance due to x1.
- x12 - joint x1-x2 effects; the species variance that is jointly explained by x1 and x2 constraints; i.e., the confounded variance that cannot exclusively be associated with either x1 or x2.

In addition, the remaining "unexplained" variance is usually referred to as the "residual" variance and is equal to the sum of the unconstrained eigenvalues. If a "conditional" data set was provided, the explained variance, partitioned into the three components above, and the residual variance are based on the variance in the species data after accounting for that due to the conditional data set.

A three-way partitioning (involving three explanatory data sets) has seven components.

- x1 - pure x1 effects; the species variance explained due to x1 constraints alone, independent of the variance due to x2 and x3.
- x2 - pure x2 effects; the species variance explained due to x2 constraints alone, independent of the variance due to x1 and x3.
- x3 - pure x2 effects; the species variance explained due to x2 constraints alone, independent of the variance due to x1 and x2.
- x12 - joint x1-x2 effects; the species variance that is jointly explained by x1 and x2 constraints (i.e., the confounded variance that cannot exclusively be associated with either x1 or x2) but independent of x3.
- x13 - joint x1-x3 effects; the species variance that is jointly explained by x1 and x3 constraints (i.e., the confounded variance that cannot exclusively be associated with either x1 or x3) but independent of x2.
- x23 - joint x3-x3 effects; the species variance that is jointly explained by x2 and x3 constraints (i.e., the confounded variance that cannot exclusively be associated with either x2 or x3) but independent of x1.
- x123 - joint x1-x2-x3 effects; the species variance that is jointly explained by x1, x2 and x3 constraints (i.e., the confounded variance that cannot exclusively be associated with either x1, x2 or x3).

Similar to above, the remaining "unexplained" variance is the "residual" variance and is equal to the sum of the unconstrained eigenvalues. If a "conditional" data set was provided, the explained variance, partitioned into the seven components above, and the residual variance are based on the variance in the species data after accounting for that due to the conditional data set.

In the component data table, the third column reports the partition sizes in terms of the proportion of the total *explained* (or constrained) variance instead of the proportion of the total species variance.

Note, in either case, the component effect size for the joint or overlapping components (e.g., v12, v13, v23, v123) can be negative if the effect of the variables together is stronger than sum of both separately. In general, such components of "variance"' are not to be trusted due to interactions between two sets of variables.

The final column in the component data table reports the results of a Monte carlo test of significance of the component effect size similar to that described

above for the marginal effects. However, a test is only possible for components that can be expressed as a partial constrained ordination model. Thus, only the conditional effects of the explanatory variable subsets can be tested. The components representing the joint or overlapping regions of the components cannot be tested directly because their effect size is determined algebraically and not by a direct model.

Author:

K. McGarigal, December 5, 2006

References:

Anderson, M.J. and N.A. Gribble. 1998. Partitioning the variation among spatial, temporal and environmental components in a multivariate data set. *Australian Journal of Ecology* 23: 158-167.

Borcard, D. and P. Legendre. 1994. Environmental control and spatial structure in ecological communities: an example using oribatid mites (Acari, Orbiatei). *Environ. Ecol. Stat.* 1:37-53.

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

Cushman, S. A., and K. McGarigal. 2004. Hierarchical analysis of forest bird species-environment relationships in the Oregon Coast Range. Ecological Applications 14:1090-1105.

Legendre, P. and D. Borcard. 1994. Rejoiner. *Environ. Ecol. Stat.* 1:57-61.

Cushman, S. A., and K. McGarigal. 2002. Hierarchical, multi-scale decomposition of species-environment relationships. Landscape Ecology 17:637-646.

Liu, Q.H. and S. Brakenhielm. 1995. A statistical approach to decompose ecological variation. *Water, Air, and Soil Pollution* (1-4): 61-87.

Smouse, P. E., J. C. Long, and R. R. Sokal. 1986. Multiple regression and correlation extensions of the Mantel test of matrix correspondence. Systematic Zoology 35:627-632.

Examples:

```
moths<-read.csv('moths.csv',header=TRUE)
hab.plot<-read.csv('moths.plot.csv',header=TRUE)
hab.patch<-read.csv('moths.patch.csv',header=TRUE)
hab.land<-read.csv('moths.land.csv',header=TRUE)
hab.space<-read.csv('moths.space.csv',header=TRUE)
y<-moths[,-1]
x.plot<-hab.plot[,c(2,9,10)]
x.patch<-hab.patch[,c(4,6,7)]
x.land<-hab.land[,c(3,8,9)]
x.space<-hab.space[,c(3,7,8)]
y.log<-data.trans(y,method='log',plot=FALSE)
```

```
y.chord<-data.stand(y.log,method='normalize',margin='row',plot=FALSE)
z<-ordi.part(y.chord,x.plot,x.patch,x.land,p=x.space,method='rda')
```

Function: ordi.scree

Description:

Scree plot of eigenvalues from eigenvector ordinations (see nmds.scree for stress-based scree plot for nonmetric multidimensional scaling). Currently, scree plots can be produced from ordination objects containing results from several different eigenvector ordination methods (see details below). In all cases, the scree plot depicts the eigenvalues in descending order. In the case of principal components derived from a correlation matrix (using either the prcomp(), princomp() or rda() functions), the scree plot also depicts the *broken stick distribution*, which is based on the expected distribution of eigenvalues when the total variance is distributed randomly among components, and the *latent root criterion*, which is based on the uniform distribution of variance among eigenvalues (i.e., eigenvalue = 1). A second plot shows the *cumulative proportion of variance or inertia*, as appropriate, accounted for by each eigenvalue.

Usage:

ordi.scree(x, ord, ...)

Dependency:

library(vegan)

Arguments:

x: *required* ordination object (of class = 'prcomp', 'princomp', 'rda', 'cca', or 'capscale'.

ord: *required* ordination method: 'pca', 'ca', 'mds', 'rda', 'cca', 'cmds' (see details below).

... *optional* additional graphics arguments passed to the plot() function.

Details:

A scree plot is a standard graphical display of the eigenvalues which is often used to help determine the appropriate number of ordination axes to retain for interpretation. The scree plot typically declines rapidly after the first eigenvalue and eventually levels off. The first major slope break is often considered the maximum number of eigenvalues to retain. For principal components analysis (PCA) based on the correlation matrix, the broken stick distribution represents the distribution of eigenvalues under the broken stick model, which assumes that the total variance is randomly divided among the *P* components. Eigenvalues above the broken stick line are deemed "significant".

Currently, ordi.scree() produces scree plots for the following methods:

pca: Principal Components Analysis (PCA) or partial Principal Components Analysis (pPCA) of either a covariance or correlation matrix computed using functions prcomp(), princomp() or rda(). The eigenvalues plotted are the *unconstrained* eigenvalues of the covariance or correlation matrix. Note, if the ordination was based on the correlation matrix (i.e., column centered and standardized data), the broken-stick and latent root criteria are also plotted.

rda: Redundancy Analysis (RDA) (or constrained Principal Components Analysis) or partial Redundancy Analysis (pRDA) of either a covariance or correlation matrix computed using the rda() function. The eigenvalues plotted are the *constrained* eigenvalues of the data matrix.

ca: Correspondence Analysis (CA, also Reciprocal Averaging, RA) or partial Correspondence Analysis (pCCA) of a community data matrix computed using the cca() function (but without supplying a constraints matrix). The eigenvalues plotted are the *unconstrained* eigenvalues of the community matrix.

cca: Canonical Correspondence Analysis (CCA, or constrained Correspondence Analysis) or partial Canonical Correspondence Analysis (pCCA) of a community data matrix computed using the cca() function. The eigenvalues plotted are the *constrained* eigenvalues of the data matrix.

mds: Multidimensional Scaling (MDS, also Principal Coordinates Analysis) of a data matrix computed using the cmdscale() function. The eigenvalues plotted are the *unconstrained* eigenvalues of the data matrix.

cmds: Constrained Multidimensional Scaling (CMDS, or constrained Principal Coordinates Analysis) or partial Constrained Multidimensional Scaling (pCMDS) of a data matrix computed using the capscale() function. The eigenvalues plotted are the *constrained* eigenvalues of the data matrix.

Value:
None.

Author:
K. McGarigal, October 7, 2006

References:
None.

Examples:
```
x<-prcomp(rip.bird, scale=TRUE)
ordi.scree(x,ord='pca')
```

Function: pca.communality

Description:

Computes final communality estimates for variables in a principal components analysis. Final communalities are the sum of the squared structure correlations (loadings) and represent the percentage of a variable's variance that is linearly accounted for (explained) by the retained principal components.

Usage:

pca.communality(x.pca, x, dim=length(x.pca$sdev), digits=3)

Dependency:

none.

Arguments:

x.pca: *required* object of class prcomp, i.e., the result of prcomp() applied to data frame containing one or more numeric variables.

dim: *optional* number of dimensions or principal components to retain. Note, retaining all principal components will result in final communality estimates of 1 for each variable, since 100% of a variable's variance is accounted for by all *P* principal components, where *P* equals the number of variables in the data matrix. [default = *P*]

digits: *optional* number of significant digits to report for the communality estimates. [default = 3]

Details:

Final communality estimates are simply squared correlation coefficients representing the sum of the squared structure coefficients (correlations) for each variable across the retained principal components. Final communalities increase monotonically with each additional retained principal component and equal 1 when all components are retained. Final communalities provide an indication of how well each variable is capture by the ordination solution. However, because communalities are based on linear correlation coefficients, they only account for the linear relationships with the ordination axes. A variable (e.g., species) with a strong unimodal relationship to the ordination axes is likely to have a small final communality even though the patterning is quite strong.

Value:

Returns a data frame or vector containing the final communalities for the *P* variables.

Author:

K. McGarigal, September 26, 2006

References:

None.

Examples:

```
x<-prcomp(rip.bird,scale=TRUE)
pca.communality(x,rip.bird,dim=5)
```

Function: pca.eigenval

Description:

Summary of the eigenvalues and associated statistics derived from a principal components analysis using the prcomp() function. In addition to the eigenvalues, the proportion of total variance accounted for by each eigenvalue, the cumulative proportion of variance accounted for by each eigenvalue, and the broken stick value for each component (i.e., the expected eigenvalue if the total variance was randomly distributed among components) are output in the summary.

Usage:

pca.eigenval(x.pca, dim=length(x.pca$sdev), digits=7)

Dependency:

none.

Arguments:

x.pca: *required* object of class prcomp, i.e., the result of prcomp() applied to data frame containing one or more numeric variables.

dim: *optional* number of dimensions or eigenvalues to print in the summary. [default = all *P* dimensions]

digits: *optional* number of significant digits to report for the eigenvalues and associates statistics. [default = 7]

Details:

The eigenvalues are derived from a singular value decomposition of the centered and possibly scaled (standardized) data matrix using the prcomp() function. The eigenvalues are equal to the squared singular values and equal the variances of the corresponding principal components. The eigenvalue divided by the sum of all *P* eigenvalues equals the proportion of variance associated with each component. Eigenvalues less than one account for less variance than a single original variable when the decomposition is based on the correlation matrix. The broken stick values are based on the expected eigenvalue distribution if the total variance is random distributed among components. Eigenvalues larger than the corresponding broken stick value are deemed "significant".

Value:

Returns a data frame containing the eigenvalue summary.

Author:

K. McGarigal, September 26, 2006

References:

None.

Examples:

```
x<-prcomp(rip.bird, scale=TRUE)
pca.eigenval(x, dim=5)
```

Function: pca.eigenvec

Description:
Summary of the eigenvectors (or variable loadings) derived from a principal components analysis using the prcomp() function.

Usage:
pca.eigenvec(x.pca, dim=length(x.pca$sdev), digits=7, cutoff=0)

Dependency:
none.

Arguments:

x.pca:	*required* object of class prcomp, i.e., the result of prcomp() applied to data frame containing one or more numeric variables.
dim:	*optional* number of dimensions or eigenvectors to print in the summary. [default = all *P* dimensions]
digits:	*optional* number of significant digits to report for the eigenvectors. [default = 7]
cutoff:	*optional* minimum eigenvector coefficient for suppressing the printing of small coefficients (loadings). [default = 0]

Details:
The eigenvectors are the weights of the variables in the linear equations that define the principal components. Each of the *P* original variables has a weight (or loading) on each principal component. The eigenvector coefficients represent the strength and direction of the relationship between each variable and each principal component. If the eigenvectors are derived from the correlation matrix, they are *proportional* to the correlations between the original variables and the ordination axes, but they are NOT correlation coefficients. Nevertheless, large positive values indicate that samples positioned on the positive end of the ordination axis contain larger values of the corresponding variable, and vice versa for negative values.

Note, it is often the case that eigenvector coefficients from two different computers or programs will come out with the same values but with opposite signs. The orientation of eigenvectors is arbitrary, and the sign is only meaningful with respect to other values on the same axis.

Value:
Returns a data frame containing the eigenvectors.

Author:
K. McGarigal, September 26, 2006

References:

None.

Examples:

```
x<-prcomp(rip.bird, scale=TRUE)
pca.eigenvec(x, dim=5, digits=3, cutoff=.1)
```

Function: pca.structure

Description:

Structure coefficients (also referred to as "structure correlations" or "correlation loadings" or just plain "loadings) for the corresponding eigenvectors derived from prcomp(). Structure coefficients simple linear correlation coefficients between the original variables and the principal components. The squared structure coefficients give the percentage of variance in each original variable accounted for by each principal component. Note that these are linear correlations, which can be misleading if the relationships are non-linear.

Usage:

pca.structure(x.pca, x, dim=length(x.pca$sdev), digits=3, cutoff=0)

Dependency:

None.

Arguments:

x.pca: *required* object of class prcomp, i.e., the result of prcomp() applied to data frame containing one or more numeric variables.

x: *required* name of data frame containing one or more numeric variables; actually, it must be the same set of variables used to derive the PCA.

dim: *optional* number of dimensions to print. [default = all *P* dimensions]

digits: *optional* number of significant digits to report for the structure coefficients. [default = 3]

cutoff: *optional* minimum structure coefficient for suppressing the printing of small coefficients (loadings). [default = 0]

Details:

Structure coefficients can be computed in different ways. They can either be computed directly from the eigenvector coefficients or they can be calculated directly as the Pearson product-moment bivariate correlation between each variable and the principal component scores for each component. Pca.structure() adopts the latter approach.

In contrast to the eigenvector coefficients, the structure coefficients are correlations and directly measure the strength of the linear relationship between each variable and each principal component. Large positive values indicate that samples positioned on the positive end of the ordination axis contain larger values of the corresponding variable, and vice versa for negative values. Structure coefficients are often as the basis for interpreting the meaning of the principal components.

Value:
A data frame containing the structure coefficients.

Author:
K. McGarigal, September 26, 2006

References:
None.

Examples:
```
x<-prcomp(rip.bird, scale=TRUE)
pca.structure(x.pca, rip.bird, dim=3, cutoff=.3)
```

Function: plot.anosim

Description:

Produces two plots from an 'anosim' object (i.e., the result of anosim() in the vegan library), including a grouped box-and-whisker plot of the between- and within-group dissimilarities and a histogram of the permuted values of the test statistic, R.

Usage:

plot.anosim(x, title1='ANOSIM (within- vs between-group rank dissimilarities)', title2='ANOSIM (observed vs expected R)', col='blue', ...)

Dependency:

None.

Arguments:

x:	*required* anosim object; the result of the anosim() function in the vegan library.
title1:	*optional* title for the first plot, the box-and-whisker plot of between- and within-group dissimilarities. [default = 'ANOSIM (within- vs between-group rank dissimilarities)']
title2:	*optional* title for the second plot, the histogram of the permuted values of the test statistic, R. [default = 'ANOSIM (observed vs expected R)']
...	*optional* additional arguments to be passed to the boxplot and hist functions, including the following defaults: col='blue' color of box-and-whiskers and histogram

Details:

plot.anosim is simply a convenience wrapper for the boxplot() and hist() functions that makes it efficient to quickly produce a grouped box-and-whisker plot and histogram of the results of ANOSIM.

Value:

No object is returned.

Author:

K. McGarigal, October 25, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
y<-turtle[,6:30]
grp<-turtle[,3]
```

```
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.anosim<-anosim(y.eucl,grp)
plot.anosim(y.anosim)
```

Function: plot.mantel

Description:

Produces two plots from a 'mantel' object (i.e., the result of mantel2() in BIOSTATS), including a scatterplot of the two input dissimilarity matrices and a histogram of the permuted values of the test statistic, r.

Usage:

plot.mantel(x, title1='MANTEL Scatterplot)', title2='MANTEL (observed vs expected R)', col='blue', ...)

Dependency:

None.

Arguments:

x:	*required* mantel object; the result of the mantel2() function in BIOSTATS.
title1:	*optional* title for the first plot, the scatterplot of the two dissimilarity matrices. [default = 'MANTEL Scatterplot)']
title2:	*optional* title for the second plot, the histogram of the permuted values of the test statistic, r. [default = 'MANTEL (observed vs expected R)']
...	*optional* additional arguments to be passed to the plot and hist functions, including the following defaults: col='blue' color of points and histogram

Details:

plot.mantel is simply a convenience wrapper for the plot() and hist() functions that makes it efficient to quickly produce a scatterplot and histogram of the results of MANTEL.

Value:

No object is returned.

Author:

K. McGarigal, October 25, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
y<-turtle[,6:30]
grp<-turtle[,3]
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
```

```
y.eucl<-data.dist(y.std,method='euclidean')
contrast<-contrast.matrix(grp)
y.mantel<-mantel2(contrast,y.eucl)
plot.mantel(y.mantel)
```

Function: plot.mrpp

Description:

Produces two plots from an 'mrpp' object (i.e., the result of mrpp() in the vegan library or mrpp2() in BIOSTATS), including a grouped box-and-whisker plot of the between- and within-group dissimilarities and a histogram of the permuted values of the test statistic, delta.

Usage:

plot.mrpp(x, title1='MRPP (within- vs between-group dissimilarities)', title2='MRPP (observed vs expected delta)', col='blue', ...)

Dependency:

None.

Arguments:

x: *required* mantel object; the result of the mantel() function in the vegan library or mantel2() function in BIOSTATS.

title1: *optional* title for the first plot, the box-and-whisker plot of between- and within-group dissimilarities. [default = 'MRPP (within- vs between-group dissimilarities)']

title2: *optional* title for the second plot, the histogram of the permuted values of the test statistic, R. [default = 'MRPP (observed vs expected delta)']

... *optional* additional arguments to be passed to the boxplot and hist functions, including the following defaults:
col='blue' color of box-and-whiskers and histogram

Details:

plot.mantel is simply a convenience wrapper for the boxplot() and hist() functions that makes it efficient to quickly produce a grouped box-and-whisker plot and histogram of the results of MANTEL.

Value:

No object is returned.

Author:

K. McGarigal, October 25, 2006

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
y<-turtle[,6:30]
```

```
grp<-turtle[,3]
y.std<-data.stand(y,method='standardize',margin='column',plot=FALSE)
y.eucl<-data.dist(y.std,method='euclidean')
y.mrpp2<-mrpp2(y.eucl,grp)
plot.mrpp(y.mrpp2)
```

Function: plot.ordi.part

Description:

This function plots the results of a variance decomposition or partitioning of a constrained ordination using the ordi.part() function in BIOSTATS. Specifically, this function produces a Venn diagram depicting the components of the partition and their effect sizes.

Usage:

plot.ordi.part(x, which='total', digits=1, ...)

Dependency:

None.

Arguments:

x: *required* ordi.part object, the result of ordi.part().

which: *optional* choice of whether to plot the component effect sizes as the percentage of the total y-set variance explained (which = 'total') or as the percentage of the total constrained or explained variance (which = 'constrained'). [default = 'total']

digits: *optional* number of digits to report in the output. [default = 1]

... *optional* additional graphical arguments to be passed to the text() function for example to control text size and color.

Details:

The Venn diagram depicts the components of a two- or three-way partition of a constrained ordination, in addition to the so-called "residual" variance and "conditional" variance if provided. The components can be displayed either as the percentage of the *total y-set variance* accounted for (which='total') or the percentage of the *total explained variance* accounted for (which='constrained'). If a "conditional" data set is provided, the Venn diagram includes a description of the percent of species variance accounted for by the conditional data set (reported in the lower left corner of the plot). In this case, all other reported components reflect the percentage of species variance accounted for by each component independent of that due to the conditional variables. The "residual" or unexplained variance is also reported in the lower right corner of the plot. This is equal to the percentage of the species variance that is accounted for by the sum of the unconstrained eigenvalues; i.e., that not due to the explanatory variables.

Value:

Returns the Venn diagram to the graphics device.

Author:

K. McGarigal, December 5, 2006

References:

None.

Examples:

```
moths<-read.csv('moths.csv',header=TRUE)
hab.plot<-read.csv('moths.plot.csv',header=TRUE)
hab.patch<-read.csv('moths.patch.csv',header=TRUE)
hab.land<-read.csv('moths.land.csv',header=TRUE)
hab.space<-read.csv('moths.space.csv',header=TRUE)
y<-moths[,-1]
x.plot<-hab.plot[,c(2,9,10)]
x.patch<-hab.patch[,c(4,6,7)]
x.land<-hab.land[,c(3,8,9)]
x.space<-hab.space[,c(3,7,8)]
y.log<-data.trans(y,method='log',plot=FALSE)
y.chord<-data.stand(y.log,method='normalize',margin='row',plot=FALSE)
z<-ordi.part(y.chord,x.plot,x.patch,x.land,p=x.space,method='rda')
plot.ordi.part(z,which='total')
```

Function: qqnorm.plots

Description:

Produces normal quantile-quantile plots for individual variables (columns) of a data frame. If a grouping variable is specified, box-and-whisker plots for each group are displayed side-by-side on the same page for efficient comparison.

Usage:

qqnorm.plots(x, var=' ', by=' ', save.plot=FALSE, na.rm=TRUE, ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]

save.plot: *optional* logical (TRUE or FALSE) to automatically save all plots as jpegs with the following naming convention: 'qqnorm.var.jpg', where 'var' is variable name. [default = FALSE]

na.rm: *optional* logical (TRUE or FALSE) whether to ignore missing values. [default=TRUE]

... *optional* additional arguments to be passed to the qqnorm() and qqline() functions, including the following defaults:

col.point='blue'	color of qqnorm points
col.line='red'	color of the qqline
las=1	horizontal orientation of axis labels

Details:

qqnorm.plots is simply a convenience wrapper for the qqnorm() and qqline() functions that makes it efficient to quickly produce normal quantile-quantile plots for many variables, and optionally by a grouping variable. See help(qqplot) for details on qqnorm() and qqline(). Note, a qqline (line project through the 1st and 3rd quartiles of the data) will be added to the plot if and only if the inter-quartile range is >0.

Value:

No object is returned.

Author:
K. McGarigal, September 14, 2006

References:
None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
qqnorm.plots(x, 'AMGO:WWPE', c('BASIN','SUB'), FALSE, TRUE)
```

Function: ran.split

Description:

Generic function for randomly splitting a grouped data set into two sets, typically a training (or calibration) set and a validation (or testing) set for use in discriminant analysis/classification.

Usage:

ran.split(x, grouping=' ', prop=.5)

Dependency:

none.

Arguments:

x: *required* numeric vector of data values, or data frame or matrix containing one or more numeric variables and optionally a grouping variable (either numeric or character) in the first column of the data frame or matrix.

grouping: *optional* vector (either numeric or character) containing the group membership of each observation in x. If omitted, the grouping variable is assumed to be the first variable (column) in x.

prop: *optional* number between 0-1 specifying the proportion of observations to be placed in the training dataset. The remaining observations are placed in the validation dataset. [default = .5]

Details:

Observations in the original data set are randomly subdivided into training and validation data sets. There is no provision (currently) for stratified random subsetting to ensure that proportional group sample sizes are preserved in the training and validation data sets. Consequently, ran.split() will produce a different outcome each time and the proportion of observations in each group in each subset will vary somewhat. With large data sets the departure from initial proportions should be relatively minor.

Value:

Returns a list with five components. The first component is a data frame containing the number of samples and the proportion of samples in the training and validation datasets. The second and third components are tables containing the frequencies in each group for the training and validation data sets, respectively. The fourth and fifth components contain the training and validation datasets, respectively.

Also returns two objects: training.grp and validate.grp which contain the group membership of each observation in the training and validation data sets, respectively. These are created to facilitate use of the training and validation data sets in subsequent calls to lda() and qda().

Author:

K. McGarigal, 4 November 2006; updated 2 February 2013

References:

None.

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
ran.split(y,grp,prop=.5)
```

Function: redun.plot

Description:

Generic function for assessing the degree of "true" redundancy in a data matrix, typically used to assess the degree of redundancy in a data matrix prior to ordination.

Usage:

redun.plot(x, var=' ', ...)

Dependency:

none.

Arguments:

x:	*required* data frame or matrix containing one or more numeric variables.
var:	*optional* list of one or more numeric variables, e.g., 'var1' or 'var1:var5'. If specified, a separate plot is produced for each variable depicting the ecdf for the correlations involving that species. If omitted, the x object must contain all numeric variables and the plot is for pairwise correlations between all variables.
perm:	*optional* integer number of random permutations. [default = 1000]
quantiles:	*optional* vector containing the lower and upper quantiles of the random permutation distribution to plot. [default = c(.025,.975)]
...	*optional* additional arguments to be passed to the plot() function.

Details:

Creates a plot depicting the actual redundancy versus that expected by chance. Specifically, the lower triangle of a correlation matrix depicting the unique pairwise correlations among the variables is rank ordered from the largest negative correlation to the largest positive correlation and then plotted. The same empirical cumulative distribution (ecdf) function is plotted for the data set after random permutation of the columns (variables) of the data matrix. More specifically, the specified lower (default = .025) and upper (default = .975) quantiles of the random permutation ecdf is plotted, by default producing a 95 percentile envelope about the null distribution of rank order correlations. A comparison of the two distributions reveals the degree of "true" redundancy. Specifically, the observed ecdf falling outside the random permutation envelope represents "true" redundancies; i.e., higher correlations than would be expected by chance. If specific variables are listed (var =), then a separate ecdf for the correlations involving the corresponding variable is produced.

Value:

Plot of the ecdf's. The x-axis is the rank order of correlations from the largest negative correlation to the largest positive correlation. The y-axis is the correlation and ranges from -1 to 1.

Author:
K. McGarigal, November 28, 2006; modified February 13, 2008

References:
None.

Examples:
```
moths<-read.csv('moths.csv',header=TRUE)
redun.plot(moths, var='ITAME:EUMA')
```

Function: replace.missing

Description:
Replaces missing values with median or mean of column.

Usage:
replace.missing<-function(x, var=' ', method='median', outfile=' ')

Dependency:
None.

Arguments:
x: *required* name of data frame containing one or more numeric variables with missing data (NA).

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

method: *optional* method of missing values replacement: 'median'or 'mean of column. [default = 'median']

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

Details:
None.

Value:
A new *data frame* with missing values replaced.

Author:
K. McGarigal, September 14, 2006

References:
None.

Examples:
x<-read.csv('testbird.csv', header=TRUE)
replace.missing(x, 'AMGO:WWPE', method='mean', 'D:/R/stats/testout')

Function: scatter.plots

Description:
Produces a separate scatterplot for each unique combination of x and y variables with an optional Lowess regression line.

Usage:
scatter.plots(data, y=' ', x=' ', fit='lowess', ...)

Dependency:
None.

Arguments:

data:	*required* name of data frame containing two or more numeric variables.
y:	*required* list of one or more numeric variables to treat as dependent variables (y's), e.g., 'var1' or 'var1:var5'.
x:	*required* list of one or more numeric variables to treat as independent variables (x's), e.g., 'var1' or 'var1:var5'.
fit:	*optional* choice of a lowess or a linear regression fit to the scatterplot: 'lowess', 'linear', or '' for no fitted line. [default = lowess]
...	*optional* additional arguments to be passed to the plot() function, including the following defaults: col.line='red' color of the lowess regression line cex.main=2 character size of the main title of the plot

Details:
scatter.plots is simply a convenience wrapper for the plot() function that makes it efficient to quickly produce scatterplots for many x-y variables. See help(plot) for details on plot() and lowess() for details on lowess regression.

Value:
No object is returned.

Author:
K. McGarigal, February 13, 2008

References:
None.

Examples:
x<-read.csv('testbird.csv', header=TRUE)

```
scatter.plots(data=x, y='AMGO', x='AMRO:WWPE')
```

Function: sum.stats

Description:

Computes a variety of row or column summary statistics for numeric variables in a data frame, including several statistics appropriate for summarizing a community data set consisting of species abundances.

Usage:

sum.stats(x, var=' ', by=' ', margin='column', outfile=' ', ...)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

var: *optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.

by: *optional* vector of grouping variables to use for column summary, e.g., c('var1','var2',...). Note, grouping variables only effect column summaries; they are ignored for row summaries. [default = no groups]

margin: *optional* choice of column or row summary. [default = 'column']

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

Details:

sum.stats computes the following row or column summary statistics for the selected variables (columns), and ignores missing values (na.rm=TRUE), except where noted:

nobs:	number of observations (including missing values)
min:	minimum value
max:	maximum value
mean:	average value
median:	median value (i.e., 50th percentile)
sum:	sum of all values
sd:	sample standard deviation
cv:	coefficient of variation (i.e., 100*sd/mean)
zeros:	number of elements with the value zero
pct.zeros:	percent of observation with the value zero
nobs.missing:	number of missing observations (NA)

pct.missing: percent of observations with missing values (NA)
se: standard error (i.e., sd/sqrt(non-missing obs))
se.ratio: standard error ratio (i.e., 100*se/mean)
richness: number of non-zero elements

sh.diversity: Shannon's diversity index:

$$D_{sh} = -\sum_{i=1}^{p} P_i \cdot \ln(P_i)$$

where P_i equals the i^{th} element of the row or column divided by the corresponding row or column sum and i is from 1 to p row or column elements.

sh.evenness: Shannon's evenness index:

$$E_{sh} = \frac{D_{sh}}{\ln(m)}$$

where m is the richness (i.e., number of non-zero elements).

si.diversity: Simpson's diversity index:

$$D_{si} = 1 - \sum_{i=1}^{p} P_i^2$$

si.evenness: Simpson's evenness index:

$$E_{si} = \frac{D_{si}}{1 - \left(1/m\right)}$$

If a 'by' argument is given, these summary statistics are computed for each level of the 'by' variable(s).

In addition to the above row/column stats, a statistical summary of the row/columns summary stats is also provided (for each group if a 'by' argument is given). Specifically, the number observations (nobs), minimum (min), maximum (max), median (median), standard deviation (sd) and coefficient of variation (cv) of the row/column summary statistics are reported.

Value:

A *list* containing two components:

Row.summary or Column.summary: Data frame with the row or column summary statistics.

Table.summary: Data frame with the statistical summary of the row or column summary statistics.

Author:

K. McGarigal, September 14, 2006

References:
None.

Examples:
```
x<-read.csv('testbird.csv', header=TRUE)
sum.stats(x, 'AMGO:WWPE', c('BASIN','SUB'), 'row', 'D:/stats/testout')
```

Function: tau

Description:

Computes the *Tau* statistic (Klecka 1980) from a classification table, typically the result of a classification of samples into groups using the predict() function on an object from linear (lda) or quadratic discrimination (qda). *Tau* is a chance-corrected measure of classification accuracy and is suitable when the prior probabilities of group membership are known or not assumed to be proportional to group sample sizes. In the latter case, Cohen's *Kappa* is generally the statistic of choice (see cohen.kappa()).

Usage:

tau(y, prior)

Dependency:

None.

Arguments:

y: *required* name of a table (class = 'table') containing the frequency of observations in each group (rows) classified into each group (columns). The table must be square and contain a single row and column for each group.

prior: *required* vector of prior probabilities of class membership. The probabilities should be specified in the order of the factor levels.

Details:

Tau is defined as:

$$Tau = \frac{n_o - \sum_{i=1}^{G} p_i n_i}{n - \sum_{i=1}^{G} p_i n_i}$$

where G is the number of groups, n is the total number of samples, n_o is the number of samples correctly classified, n_i is the number of samples in the i^{th} group, and p_i is the prior probability of membership in the i^{th} group

The term involving the summation is the number of samples that would be correctly classified on the basis of random assignment to groups in proportion to their prior probabilities. If the groups are to be treated equally, then all the prior probabilities are set to one divided by the number of groups. The maximum value for *Tau* is one and it occurs when there are no errors in prediction. The minimum value of *Tau* is zero and it indicates no improvement over chance. An intermediate value of *Tau* such as 0.82, for example, indicates that classification based on the discriminating variables made 82% fewer errors than would be expected by random assignment.

It is important to note that *Tau*, like any measure of classification success, is unbiased only when

computed with 'holdout' samples. In other words, for unbiased results, the accuracy of the classification criterion should be evaluated by comparing the classification results and chance-corrected criteria computed from a 'holdout' or 'validation' sample. This is because the classification functions are more accurate for the samples they are derived from than they would be for the full population. Thus, if the samples used in calculating the classification function are the ones being classified, the result will be an upward bias in the correct classification rate.

Value:

Returns an object containing the value of *Tau*.

Author:

K. McGarigal, November 4, 2006

References:

Klecka, W. R. 1980. *Discriminant Analaysis*. Sage University Paper series Quantitative Applications in the Social Sciences, series no. 07-019. Beverly Hills and London: Sage Publications..

Examples:

```
turtle<-read.csv('byturtle.csv',header=TRUE)
grp<-turtle[,3]
y<-turtle[,6:30]
y.qda<-qda(y,grouping=grp)
y.qda.pred<-predict(y.qda)
y.table<-table(grp,y.qda.pred$class)
tau(y.table,prior=y.qda$prior) #priors proportional to group sample sizes
tau(y.table,prior=c(.2,.8)) #priors specified
```

Function: uv.outliers

Description:

Screens numeric data in a data frame for potential *univariate* "outliers." More accurately, uv.outliers calculates the z-scores via a column standardization (i.e., adjusting the scores in each column to have zero mean and unit variance) and then returns a list of the samples (rows) and the corresponding z-scores that exceed a threshold level of standard deviations from the mean. Optionally, extreme values can be computed separately for each group of samples based on one or more grouping variables.

Usage:

univar.outliers(x, id=, var=, by=NULL, outfile=NULL, sd.limit=3)

Dependency:

None.

Arguments:

x: *required* name of data frame containing one or more numeric variables.

id: *required* list of one or more variables to retain in the output, for example, to identify the record, e.g., 'var1' or 'var1:var5'.

var: *required* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'.

by: *optional* vector of grouping variables to use for grouping records, e.g., c('var1','var2',...). Note, standardization will occur within each group of records. [default = no groups]

outfile: *optional* name of an output file in comma-delimited format, e.g., 'testout' or 'D:/R/work/testout'. The output file will automatically be given a .csv extension. Note, path does not need to be included if the desired output location is the current working directory. [default = no output file]

sd.limit *optional* threshold level in standard deviations for flagging extreme values. [default = 3]

Details:

Extreme values can exert undue influence on the results of most multivariate techniques. Thus, it is good practice to screen the data for extreme values before conducting any analysis. There are many ways to identify extreme observations. Univar.outliers identifies samples (rows) that have extreme values on one or more independent variables (columns); consequently, this is a "univariate" method of identifying extreme values. By specifying one or more grouping variables, you can screen for extreme values within each group of observations. Importantly, just because an observation is extreme, doesn't mean that it should automatically be deemed an "outlier" and deleted from the data set. In general, no observation should be dropped unless it can be justified on ecological grounds, e.g., that it represents a real ecological oddity.

Value:

Returns a new *data frame* containing the reduced set of samples (rows) and variables (columns) with extreme values.

Author:

K. McGarigal, September 14, 2006

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
uv.outliers(x, id='BLOCK ',var='AMGO:WWPE', by=c('BASIN','SUB'),
outfile='D:/stats/testout', sd.limit=3)
```

Function: uv.plots

Description:

Produces a set of four univariate plots for individual *numeric* variables (columns) of a data frame, including a histogram, box-and-whisker plot, empirical distribution plot and normal quantile-quantile plot. Each of these plots are described in more detail elsewhere (see hist.plots, box.plots, edf.plots, and qqnorm.plots, respectively). Note, this function does not produce plots for grouped data. If a grouping variable is present, then use the separate plotting functions.

Usage:

uv.plots(x, var=NULL, ...)

Dependency:

None.

Arguments:

x:	*required* name of data frame containing one or more numeric variables.
var:	*optional* list of one or more numeric variables to summarize, e.g., 'var1' or 'var1:var5'. If omitted, x object must contain all numeric variables.
...	*optional* additional arguments to be passed to the qqnorm() and qqline() functions, including the following defaults: col.fill='blue' color of the filled areas for box plot and histogram col.point='black' color of the ecdf col.line='red' color of the diagonal

Details:

uv.plots is simply a convenience wrapper for the hist(), boxplot(), plot(), qqnorm() and qqline() functions that makes it efficient to quickly produce a set of univariate summary plots for many variables. See the corresponding help files (e.g., help(qqplot)) for more details on these functions.

Value:

No object is returned.

Author:

K. McGarigal, February 11, 2008

References:

None.

Examples:

```
x<-read.csv('testbird.csv', header=TRUE)
uv.plots(x, 'AMGO:WWPE')
```