Package 'sinkr'

September 4, 2014

Type Package

Title sinkr

Version 1.0

Date 2014-07-21

Author Marc Taylor <ningkiling@gmail.com>

Maintainer Marc Taylor <ningkiling@gmail.com>

Imports irlba, vegan

Description A collection of functions featured on ``http://menugget.blogspot.com''

License MIT + file LICENSE

URL https://github.com/menugget/sinkr, http://menugget.blogspot.com

R topics documented:

---

addAlpha *Add alpha channel (transparency) to colors*

Description

Takes a vector of colors and adds an alpha channel at the given level of transparency.

Usage

```
addAlpha(COLORS, ALPHA)
```

Arguments

COLORS	Vector of any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or a positive integer i meaning palette()[i].
ALPHA	A value (between 0 and 1) indicating the alpha channel (opacity) value.

Examples

```
# Make background image
x <- seq(-180, 180,, 30)
y <- seq(-90, 90,, 30)
grd <- expand.grid(x=x,y=y)
z <- sqrt(grd$x^2+grd$y^2)
dim(z) <- c(length(x), length(y))
pal <- colorRampPalette(c(rgb(1,1,1), rgb(0,0,0)))
COLORS <- pal(20)
image(x,y,z, col=COLORS)

# Add semi-transparent layer
z2 <- grd$x^2+grd$y
dim(z2) <- c(length(x), length(y))
pal <- colorRampPalette(c(rgb(0.5,1,0), rgb(0,1,1), rgb(1,1,1)))
COLORS <- addAlpha(pal(20), 0.4) # alpha chanel equals 0.4
image(x,y,z2, col=COLORS, add=TRUE)
```

---

bioEnv *Clarke and Ainsworth's BIO-ENV routine*

Description

The bioEnv function performs Clarke and Ainsworth's (1993) "BIO-ENV" routine which compares (via a Mantel test) a fixed matrix of similarities to a variable one that test all possible variable combinations.

Usage

```
bioEnv(fix.mat, var.mat, fix.dist.method = "bray",
  var.dist.method = "euclidean", scale.fix = FALSE, scale.var = TRUE,
  output.best = 10, var.max = ncol(var.mat))
```

Arguments

`fix.mat`	The "fixed" matrix of community or environmental sample by variable values
`var.mat`	A "variable" matrix of community or environmental sample by variable values
`fix.dist.method`	The method of calculating dissimilarity indices bewteen samples in the fixed matrix (Uses the `vegdist` function from the vegan package to calculate distance matrices. See the documentation for available methods.). Defaults to Bray-Curtis dissimularity `"bray"`.
`var.dist.method`	The method of calculating dissimilarity indices bewteen samples in the variable matrix. Defaults to Euclidean dissimularity `"euclidean"`.
`scale.fix`	Logical. Should fixed matrix be centered and scaled (Defaults to `FALSE`, recommended for biologic data).
`scale.var`	Logical. Should fixed matrix be centered and scaled (Defaults to `TRUE`, recommended for environmental data to correct for differing units between variables).
`output.best`	Number of best combinations to return in the results object (Default=10).
`var.max`	Maximum number of variables to include. Defaults to all, `var.max=ncol(var.mat)`.

Details

The R package "vegan" contains a version of Clarke and Ainsworth's (1993) BIOENV analysis (`bioenv`) which allows for the comparison of distance/similarity matrices between two sets of data having either samples or variables in common. The difference with `bioEnv` is that one has more flexibility with methods to apply to the fixed and variable multivariate matrices. The typical setup is in the exploration of environmental variables that best correlate to sample similarities of the biological community (e.g. species biomass or abundance), called "BIOENV". In this case, the similarity matrix of the community is fixed, while subsets of the environmental variables are used in the calculation of the environmental similarity matrix. A correlation coefficient (typically Spearman rank correlation coefficient, "rho") is then calculated between the two matrices and the best subset of environmental variables can then be identified and further subjected to a permutation test to determine significance. The vegan package's `bioenv` function assumes BIOENV setup, and the similarity matrix of environmental data is assumed to be based on normalized "euclidean" distances. This makes sense with environmental data where one normalizes the data to remove the effect of differing units between parameters, yet in cases where the variable matrix is biological, one might want more flexibility (a Bray-Curtis measure of similarity is common given its non-parametric nature). For example, beyond the typical biological to environmental comparison (BIOENV setup), one can also use the routine to explore other other types of relationships; e.g.:

ENVBIO:	subset of biological variables that best correlate to the overall environmental pattern
BIOBIO:	subset of biological variables that best correlate to the overall biological pattern
ENVENV:	subset of environmental variables that best correlate to the overall environmental pattern

It is important to mention that one of the reasons why a variable biological similarity matrix is often less explored with the routine is that the number of possible subset combinations becomes computationally overwhelming when the number of species/groups is large - the total number of combinations being equal to 2^n - 1, where n is the total number of variables. For this reason, Clarke and Warwick (1998) presented a stepwise routine (BVSTEP) (see `bvStep` for more efficient exploration of the subset combinations).

References

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

Clarke, K. R., Warwick, R. M., 2001. Changes in Marine Communities: An Approach to Statistical Analysis and Interpretation, 2nd edition. PRIMER-E Ltd, Plymouth, UK.

Examples

```
library(vegan)
data(varespec)
data(varechem)

res <- bioEnv(wisconsin(varespec), varechem,
              fix.dist.method="bray", var.dist.method="euclidean",
              scale.fix=FALSE, scale.var=TRUE
)
res
```

---

bvStep *Clarke and Ainsworth's BVSTEP routine*

Description

The bvStep function performs Clarke and Ainsworth's (1993) "BVSTEP" routine which is a algorithm that searches for highest correlation (Mantel test) between dissimilarities of a fixed and variable multivariate datasets. The test is the same as that performed by the bioEnv function but the routine provides a more efficient search of combinations when the number of variables is large.

Usage

```
bvStep(fix.mat, var.mat, fix.dist.method = "bray",
  var.dist.method = "euclidean", scale.fix = FALSE, scale.var = TRUE,
  max.rho = 0.95, min.delta.rho = 0.001, random.selection = TRUE,
  prop.selected.var = 0.2, num.restarts = 10, var.always.include = NULL,
  var.exclude = NULL, output.best = 10)
```

Arguments

fix.mat	The "fixed" matrix of community or environmental sample by variable values
var.mat	A "variable" matrix of community or environmental sample by variable values
fix.dist.method	The method of calculating dissimilarity indices bewteen samples in the fixed matrix (Uses the vegdist function from the vegan package to calculate distance matrices. See the documentation for available methods.). Defaults to Bray-Curtis dissimularity "bray".
var.dist.method	The method of calculating dissimilarity indices bewteen samples in the variable matrix. Defaults to Euclidean dissimularity "euclidean".

`scale.fix` Logical. Should fixed matrix be centered and scaled (Defaults to `FALSE`, recommended for biologic data).

`scale.var` Logical. Should fixed matrix be centered and scaled (Defaults to `TRUE`, recommended for environmental data to correct for differing units between variables).

`max.rho` Numeric value between 0 and 1. Provides a maximum Spearman rank correlation ("rho") by which to stop the searching process. This is especially important when conducting a "BIOBIO" or "ENVENV" type setup where rho will be equal to 1 with the full set of variables (see `bioEnv` for an explanation to these types of setups). Defaults to `max.rho=0.95`

`min.delta.rho` Numeric value. Defines a minimum change in the improvement of Spearman rank correlation ("rho"). When not satisfied, `bvStep` will terminate the search process and return results of the best variable correlations.

`random.selection`
Logical. When `random.selection=TRUE` (Default), the algorithm will begin each restart with a random number of variables from the variable dataset. When `random.selection=FALSE`, a single search is conducted starting with all variables.

`prop.selected.var`
Numeric. Value between 0 and 1 indicating the proportion of variables to include at each restart.

`num.restarts` Numeric. Number of restarts (Default: `num.restarts=50`)

`var.always.include`
Numeric vector. A vector of column numbers from the variable dataset to include at the each restart.

`var.exclude` Numeric vector. A vector of column numbers from the variable dataset to always exclude at the each restart and during the search process.

`output.best` Numeric value. Number of best combinations to return in the results object (Default=10).

Details

The variable multivariate data set has 2^n-1 possible combinations to test, where n is the number of variables. Testing all variable combinations is thus unrealistic, computationally, when the number of variables is high (e.g. 20 variables contain >1e6 combinations). This may often be the case when conducting a BIOBIO type analysis , where the number of species combinations to search can be quite large (see `bioEnv` for an explanation of other types of analyses beyond the typical "BIOENV"). Below is an example of a two-step search refinement for searching for subsets of variables that best correlate with a fixed mutlivariate set.

References

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

Examples

```
library(vegan)
data(varespec)
data(varechem)
```

```
# Example of a 2-round BIO-BIO search. Uses the most frequently included variables
# in the first round at the beginning of each restart in the second round
# first round
set.seed(1)
res.biobio1 <- bvStep(wisconsin(varespec), wisconsin(varespec),
 fix.dist.method="bray", var.dist.method="bray",
 scale.fix=FALSE, scale.var=FALSE,
 max.rho=0.95, min.delta.rho=0.001,
 random.selection=TRUE,
 prop.selected.var=0.3,
 num.restarts=50,
 output.best=10,
 var.always.include=NULL
)
res.biobio1 # Best rho equals 0.833 (10 of 44 variables)

#second round - always includes variables 23, 26, and 29 ("Cla.ran" "Cla.coc" "Cla.fim")
set.seed(1)
res.biobio2  <- bvStep(wisconsin(varespec), wisconsin(varespec),
 fix.dist.method="bray", var.dist.method="bray",
 scale.fix=FALSE, scale.var=FALSE,
 max.rho=0.95, min.delta.rho=0.001,
 random.selection=TRUE,
 prop.selected.var=0.3,
 num.restarts=50,
 output.best=10,
 var.always.include=c(23,26,29)
)
res.biobio2 # Best rho equals 0.895 (15 of 44 variables)

# A plot of best variables
MDS_res=metaMDS(wisconsin(varespec), distance = "bray", k = 2, trymax = 50)
bio.keep <- as.numeric(unlist(strsplit(res.biobio2$order.by.best$var.incl[1], ",")))
bio.fit <- envfit(MDS_res, varespec[,bio.keep], perm=999)
bio.fit

plot(MDS_res$points, t="n",xlab="NMDS1", ylab="NMDS2")
plot(bio.fit, col="gray50", cex=0.8, font=4) # display only those with p>0.1
text(MDS_res$points, as.character(1:length(MDS_res$points[,1])), cex=0.7)
mtext(paste("Stress =",round(MDS_res$stress, 2)), side=3, adj=1, line=0.5)

# Display only those with envfit p >= 0.1
plot(MDS_res$points, t="n",xlab="NMDS1", ylab="NMDS2")
plot(bio.fit, col="gray50", p.max=0.1, cex=0.8, font=4) # p.max=0.1
text(MDS_res$points, as.character(1:length(MDS_res$points[,1])), cex=0.7)
mtext(paste("Stress =",round(MDS_res$stress, 2)), side=3, adj=1, line=0.5)
```

---

cov4gappy *Covariance matrix calculation for gappy data*

Description

This function calculates a covoriance matrix for data that contain missing values ('gappy data').

Usage

```
cov4gappy(F1, F2 = NULL)
```

Arguments

F1	A data field.
F2	An optional 2nd data field.

Details

This function gives comparable results to `cov(F1, y=F2, use="pairwise.complete.obs")` whereby each covariance value is divided by n number of shared values (as opposed to n-1 in the case of `cov()`. Futhermore, the function will return a 0 (zero) in cases where no shared values exist between columns; the advantage being that a covariance matrix will still be calculated in cases of very gappy data, or when spatial locations have accidentally been included without observations (i.e. land in fields of aquatic-related parameters).

Value

A matrix with covariances between columns of `F1`. If both `F1` and `F2` are provided, then the covariances between columns of `F1` and the columns of `F2` are returned.

Examples

```
# Create synthetic data
set.seed(1)
mat <- matrix(rnorm(500, sd=10), nrow=50, ncol=10)
matg <- mat
matg[sample(length(mat), 0.5*length(mat))] <- NaN # Makes 50% missing values
matg # gappy matrix

# Calculate covariance matrix and compare to 'cov' function output
c1 <- cov4gappy(matg)
c2 <- cov(matg, use="pairwise.complete.obs")
plot(c1,c2, main="covariance comparison", xlab="cov4gappy", ylab="cov")
abline(0,1,col=8)
```

---

dineof *DINEOF (Data Interpolating Empirical Orthogonal Functions)*

Description

This function is based on the DINEOF (Data Interpolating Empirical Orthogonal Functions) procedure described by Beckers and Rixon (2003). The procedure has been shown to accurately determine Emprirical Orthogonal Functions (EOFs) from gappy data sets (Taylor et al. 2013). Rather than directly return the EOFs, the results of the `dineof` function is a fully interpolated matrix which can then be subjected to a final EOF decomposition with `eof`, `prcomp`, or other EOF/PCA function of preference.

Usage

```
dineof(Xo, n.max = NULL, ref.pos = NULL, delta.rms = 1e-05)
```

Arguments

Xo	A gappy data field.
n.max	A maximum number of EOFs to iterate (leave equalling "NULL" if algorithm shold proceed until convergence)
ref.pos	A vector of non-gap reference positions by which errors will be assessed via root mean squared error ("RMS"). If ref.pos = NULL, then either 30 or 1 (which ever is larger) will be sampled at random.
delta.rms	The threshold for RMS convergence.

Value

Results of dineof are returned as a list containing the following components:

Xa	The data field with interpolated values (via EOF reconstruction) included.
n.eof	The number of EOFs used in the final solution.
RMS	A vector of the RMS values from the iteration.
NEOF	A vector of the number of EOFs used at each iteration.

References

Beckers, Jean-Marie, and M. Rixen. "EOF Calculations and Data Filling from Incomplete Oceanographic Datasets." Journal of Atmospheric and Oceanic Technology 20.12 (2003): 1839-1856.

Taylor, Marc H., Martin Losch, Manfred Wenzel, Jens Schroeter (2013). On the Sensitivity of Field Reconstruction and Prediction Using Empirical Orthogonal Functions Derived from Gappy Data. J. Climate, 26, 9194-9205.

Examples

```
# Make synthetic data field
m=50
n=100
frac.gaps <- 0.5 # the fraction of data with NaNs
N.S.ratio <- 0.1 # the Noise to Signal ratio for adding noise to data
x <- (seq(m)*2*pi)/m
t <- (seq(n)*2*pi)/n
Xt <-
 outer(sin(x), sin(t)) +
 outer(sin(2.1*x), sin(2.1*t)) +
 outer(sin(3.1*x), sin(3.1*t)) +
 outer(tanh(x), cos(t)) +
 outer(tanh(2*x), cos(2.1*t)) +
 outer(tanh(4*x), cos(0.1*t)) +
 outer(tanh(2.4*x), cos(1.1*t)) +
 tanh(outer(x, t, FUN="+")) +
 tanh(outer(x, 2*t, FUN="+")
)

# Color palette
pal <- colorRampPalette(c("blue", "cyan", "yellow", "red"))

#  The "true" fieldd
Xt <- t(Xt)
```

```
image(Xt, col=pal(100))

# The "noisy" field
set.seed(1)
RAND <- matrix(runif(length(Xt), min=-1, max=1), nrow=nrow(Xt), ncol=ncol(Xt))
R <- RAND * N.S.ratio * Xt
Xp <- Xt + R
image(Xp, col=pal(100))

# The "observed" gappy field field
set.seed(1)
gaps <- sample(seq(length(Xp)), frac.gaps*length(Xp))
Xo <- replace(Xp, gaps, NaN)
image(Xo, col=pal(100))

# The dineof "interpolated" field
set.seed(1)
RES <- dineof(Xo)
Xa <- RES$Xa
image(Xa, col=pal(100))

# Final comparison of all fields
ZLIM <- range(Xt, Xp, Xo, Xa, na.rm=TRUE)
op <- par(mfrow=c(2,2), mar=c(3,3,3,1))
image(z=Xt, zlim=ZLIM, main="A) True", col=pal(100), xaxt="n", yaxt="n", xlab="", ylab="")
box()
mtext("t", side=1, line=0.5)
mtext("x", side=2, line=0.5)
image(z=Xp, zlim=ZLIM, main=paste("B) True + Noise (N/S = ", N.S.ratio, ")", sep=""),
col=pal(100), xaxt="n", yaxt="n", xlab="", ylab="")
box()
mtext("t", side=1, line=0.5)
mtext("x", side=2, line=0.5)
box()
image(z=Xo, zlim=ZLIM, main=paste("C) Observed (", frac.gaps*100, " % gaps)", sep=""),
col=pal(100), xaxt="n", yaxt="n", xlab="", ylab="")
mtext("t", side=1, line=0.5)
mtext("x", side=2, line=0.5)
image(z=Xa, zlim=ZLIM, main="D) Reconstruction", col=pal(100), xaxt="n", yaxt="n",
xlab="", ylab="")
box()
mtext("t", side=1, line=0.5)
mtext("x", side=2, line=0.5)
par(op)
```

`earthBear` *Directional bearing between two geographic locations*

Description

`earthBear` calculates bearing (in degrees) between two lon/lat positions. One of the lon/lat positions (i.e. lon1 and lat1) can be a vector of positions to compare against the other lon/lat position (i.e. lon2 and lat2)

Usage

```
earthBear(lon1, lat1, lon2, lat2)
```

Arguments

lon1	longitude 1 (in decimal degrees)
lat1	Latitude 1 (in decimal degrees)
lon2	longitude 2 (in decimal degrees)
lat2	Latitude 2 (in decimal degrees)

Value

Vector of directional bearings (degrees)

Examples

```
earthBear(0,0,20,20)
```

---

earthDist *Earth distance between two geographic locations*

Description

earthDist calculates distance (in kilometers) between two lon/lat positions. The function assumes a mean equatorial Earth radius of 6378.145 km. One of the lon/lat positions (i.e. lon1 and lat1) can be a vector of positions to compare against the other lon/lat position (i.e. lon2 and lat2)

Usage

```
earthDist(lon1, lat1, lon2, lat2)
```

Arguments

lon1	longitude 1 (in decimal degrees)
lat1	Latitude 1 (in decimal degrees)
lon2	longitude 2 (in decimal degrees)
lat2	Latitude 2 (in decimal degrees)

Value

Vector of distances (km)

Examples

```
earthDist(0,0,20,20)
```

eof *EOF (Empirical Orthogonal Functions analysis)*

Description

This function conducts an Empirical Orthogonal Function analysis (EOF) via a covariance matrix (cov4gappy function) and is especially designed to handle gappy data (i.e. containing missing values - NaN)

Usage

```
eof(F1, centered = TRUE, scaled = FALSE, nu = NULL, method = NULL,
  recursive = FALSE)
```

Arguments

F1	A data field. The data should be arraunged as samples in the column dimension (typically each column is a time series for a spatial location).
centered	Logical (TRUE/FALSE) to define if F1 should be centered prior to the analysis. Defaults to 'TRUE'
scaled	Logical (TRUE/FALSE) to define if F1 should be scaled prior to the analysis. Defaults to 'TRUE'
nu	Numeric value. Defines the number of EOFs to return. Defaults to return the full set of EOFs.
method	Method for matrix decomposition ('svd', 'eigen', 'irlba'). Defaults to 'svd' when method = NULL. Use of 'irlba' can dramatically speed up computation time when recursive = TRUE but may produce errors in computing trailing EOFs. Therefore, this option is only advisable when the field F1 is large and when only a partial decomposition is desired (i.e. nu << dim(F1)[2]). All methods should give identical results when recursive=TRUE. svd and eigen give similar results for non-gappy fields, but will differ slightly with gappy fields due to decomposition of a nonpositive definite covariance matrix. Specifically, eigen will produce negative eigenvalues for trailing EOFs, while singular values derived from svd will be strictly positive.
recursive	Logical. When TRUE, the function follows the method of "Recursively Subtracted Empirical Orthogonal Functions" (RSEOF) (Taylor et al. 2013). RSEOF is a modification of a least squares EOF approach for gappy data (LSEOF) (see von Storch and Zwiers 1999)

Details

Taylor et al. (2013) demonstrated that the RSEOF approach more accurately estimates EOFs from a gappy field than the traditional LSEOF method. Pre-treatment of gappy fields through in EOF interpolation (dineof) may provide the most accurate EOFs, although computation time is substantially longer than RSEOF.

Value

Results of eof are returned as a list containing the following components:

u	EOFs.
Lambda	Singular values.
A	EOF coefficients (i.e. 'Principal Components').
F1_dim	Dimensions of field F1.
F1_center	Vector of center values from each column in field F1.
F1_scale	Vector of scale values from each column in field F1.

References

Bjoernsson, H. and Venegas, S.A. (1997). "A manual for EOF and SVD analyses of climate data", McGill University, CCGCR Report No. 97-1, Montreal, Quebec, 52pp.

von Storch, H, Zwiers, F.W. (1999). Statistical analysis in climate research. Cambridge University Press.

Taylor, Marc H., Martin Losch, Manfred Wenzel, Jens Schroeter (2013). On the Sensitivity of Field Reconstruction and Prediction Using Empirical Orthogonal Functions Derived from Gappy Data. J. Climate, 26, 9194-9205. pdf

Examples

```
# EOF of 'iris' dataset
Et <- eof(iris[,1:4])
plot(Et$A, col=iris$Species)

# Compare to results of 'prcomp'
Pt <- prcomp(iris[,1:4])
plot(Et$A, Pt$x) # Sign may be different

# Compare to a gappy dataset (sign of loadings may differ between methods)
iris.gappy <- as.matrix(iris[,1:4])
set.seed(1)
iris.gappy[sample(length(iris.gappy), 0.25*length(iris.gappy))] <- NaN
Eg <- eof(iris.gappy, method="svd", recursive=TRUE) # recursive ("RSEOF")
op <- par(mfrow=c(1,2))
plot(Et$A, col=iris$Species)
plot(Eg$A, col=iris$Species)
par(op)

# Compare Non-gappy vs. Gappy EOF loadings
op <- par(no.readonly = TRUE)
layout(matrix(c(1,2,1,3),2,2), widths=c(3,3), heights=c(1,4))
par(mar=c(0,0,0,0))
plot(1, t="n", axes=FALSE, ann=FALSE)
legend("center", ncol=4, legend=colnames(iris.gappy), border=1, bty="n",
fill=rainbow(4))
par(mar=c(6,3,2,1))
barplot(Et$u, beside=TRUE, col=rainbow(4), ylim=range(Et$u)*c(1.15,1.15))
mtext("Non-gappy", side=3, line=0)
axis(1, labels=paste("EOF", 1:4), at=c(3, 8, 13, 18), las=2, tick=FALSE)
barplot(Eg$u, beside=TRUE, col=rainbow(4), ylim=range(Et$u)*c(1.15,1.15))
mtext("Gappy", side=3, line=0)
axis(1, labels=paste("EOF", 1:4), at=c(3, 8, 13, 18), las=2, tick=FALSE)
par(op)
```

eofRecon *EOF reconstruction (Empirical Orthogonal Functions analysis)*

Description

This function reconstructs the original field from an EOF object of the function eof.

Usage

```
eofRecon(EOF, pcs = NULL)
```

Arguments

EOF	An object resulting from the function eof.
pcs	The principal components (PCs) to use in the reconstruction (defaults to the fullset of PCs: pcs=seq(ncol(EOF$u)))

Examples

```
set.seed(1)
iris.gappy <- as.matrix(iris[,1:4])
iris.gappy[sample(length(iris.gappy), 0.25*length(iris.gappy))] <- NaN
Er <- eof(iris.gappy, method="svd", recursive=TRUE) # recursive (RSEOF)
Enr <- eof(iris.gappy, method="svd", recursive=FALSE) # non-recursive (LSEOF)
iris.gappy.recon.r <- eofRecon(Er)
iris.gappy.recon.nr <- eofRecon(Enr)

# Reconstructed values vs. observed values
op <- par(mfrow=c(1,2))
lim <- range(iris.gappy, na.rm=TRUE)
plot(iris.gappy, iris.gappy.recon.r,
col=c(2:4)[iris$Species], main="recursive=TRUE", xlim=lim, ylim=lim)
abline(0, 1, col=1, lwd=2)
plot(iris.gappy, iris.gappy.recon.nr,
col=c(2:4)[iris$Species], main="recursive=FALSE", xlim=lim, ylim=lim)
abline(0, 1, col=1, lwd=2)
par(op)

# Reconstructed values from gappy data vs. all original values
op <- par(mfrow=c(1,2))
plot(as.matrix(iris[,1:4]), iris.gappy.recon.r,
col=c(2:4)[iris$Species], main="recursive=TRUE")
abline(0, 1, col=1, lwd=2)
plot(as.matrix(iris[,1:4]), iris.gappy.recon.nr,
col=c(2:4)[iris$Species], main="recursive=FALSE")
abline(0, 1, col=1, lwd=2)
```

expmat *Exponentiation of a matrix*

Description

The expmat function performs can calculate the pseudoinverse (i.e. "Moore-Penrose pseudoinverse") of a matrix (EXP=-1) and other exponents of matrices, such as square roots (EXP=0.5) or square root of its inverse (EXP=-0.5). The function arguments are a matrix (MAT), an exponent (EXP), and a tolerance level for non-zero singular values. The function follows three steps: 1) Singular Value Decomposition (SVD) of the matrix; 2) Exponentiation of the singular values; 3) Re-calculation of the matrix with the new singular values

Usage

```
expmat(MAT, EXP, tol = NULL)
```

Arguments

MAT	A matrix.
EXP	An exponent to apply to the matrix MAT.
tol	Tolerance level for non-zero singular values.

Value

A matrix

Examples

```
# Example matrix from Wilks (2006)
A <- matrix(c(185.47,110.84,110.84,77.58),2,2)
A
solve(A) #inverse
expmat(A, -1) # pseudoinverse
expmat(expmat(A, -1), -1) #inverse of the inverse -return to original A matrix
expmat(A, 0.5) # square root of a matrix
expmat(A, -0.5) # square root of its inverse
expmat(expmat(A, -1), 0.5) # square root of its inverse (same as above)

# Pseudoinversion of a non-square matrix
set.seed(1)
D <- matrix(round(runif(24, min=1, max=100)), 4, 6)
D
expmat(D, -1)
expmat(t(D), -1)

# Pseudoinversion of a square matrix
set.seed(1)
D <- matrix(round(runif(25, min=1, max=100)), 5, 5)
solve(D)
expmat(D, -1)
solve(t(D))
expmat(t(D), -1)
```

```
### Examples from "corpcor" package manual
# a singular matrix
m = rbind(
  c(1,2),
  c(1,2)
)

# not possible to invert exactly
# solve(m) # produces an error
p <- expmat(m, -1)

# characteristics of the pseudoinverse
zapsmall( m %*% p %*% m )  ==  zapsmall( m )
zapsmall( p %*% m %*% p )  ==  zapsmall( p )
zapsmall( p %*% m )  ==  zapsmall( t(p %*% m ) )
zapsmall( m %*% p )  ==  zapsmall( t(m %*% p ) )

# example with an invertable matrix
m2 = rbind(
  c(1,1),
  c(1,0)
)
zapsmall( solve(m2) ) == zapsmall( expmat(m2,-1) )
```

`gmtColors` *GMT palette colors*

Description

`gmtColors` provides colors used in various palettes used by Generic Mapping Tools (GMT)

Usage

```
gmtColors(pal.name = "relief")
```

Arguments

`pal.name`	A palette name - One of the following 19 palette names: ("cool", "copper", "gebco", "globe", "gray", "haxby", "hot", "jet", "no_green", "ocean", "polar", "rainbow", "red2green", "relief", "sealand","seis", "split", "topo", "wysiwyg")

Value

a vector of hexadecimal color levels of the desired palette

Examples

```
# Visualization of palettes derived from GMT colors
pnames <- c(
  "cool", "copper", "gebco", "globe", "gray", "haxby",
  "hot", "jet", "no_green", "ocean", "polar", "rainbow",
  "red2green", "relief", "sealand","seis", "split", "topo", "wysiwyg"
)
txtCol <- c(rep(1,16), "white", rep(1,3))
```

```
op <- par(mar=c(0.1,0.1,0.1,0.1), mfrow=c(length(pnames), 1))
for(i in seq(pnames)){
  pal <- colorRampPalette(gmtColors(pal.name=pnames[i]))
image(matrix(seq(20), nrow=20, ncol=1), col=pal(20), axes=FALSE)
  box()
  usr <- par()$usr
  text(mean(usr[1:2]), mean(usr[3:4]), labels=pnames[i],
       font=2, cex=1, col=txtCol[i])
}
par(op)

# Application to \code{image} plot
relief.pal <- colorRampPalette(gmtColors("relief"))
op <- par(mar=c(1,1,1,1))
image(volcano, col=relief.pal(100), axes=FALSE)
par(op)
```

`imageScale` *Make a color scale to accompany an image or other plot*

Description

The `imageScale` function is wrapper for `imageScale` and accepts the same arguments. It converts a vector of values (`z`) to a vector of color levels. One must define the number of colors. The limits of the color scale ("zlim") or the break points for the color changes("breaks") can also be defined. When breaks and zlim are defined, breaks overrides zlim. All arguments are similar to those in the `image` function. Appearance is best when incorporated with `layout`.

Usage

```
imageScale(z, zlim, col = heat.colors(12), breaks, axis.pos = 1,
  add.axis = TRUE, xlim = NULL, ylim = NULL, ...)
```

Arguments

`z`	A vector or matrix of values.
`zlim`	Limits of the color scale values.
`col`	Vector of color values (default is 12 colors from the `heat.colors` palette).
`breaks`	Break points for color changes. If breaks is specified then `zlim` is unused and the algorithm used follows `cut`, so intervals are closed on the right and open on the left except for the lowest interval which is closed at both ends.
`axis.pos`	Position of the axis (1=bottom, 2=left, 3=top, 4=right) (default = 1).
`add.axis`	Logical (TRUE/FALSE). Defines whether the axis is added (default: TRUE).
`xlim`	Limits for the x-axis.
`ylim`	Limits for the y-axis.
`...`	Additional graphical parameters to pass to the `image()` function.

Examples

```
# Make color palettes
pal.1=colorRampPalette(c("green4", "orange", "red", "white"), space="rgb", bias=0.5)
pal.2=colorRampPalette(c("blue", "cyan", "yellow", "red", "pink"), space="rgb")

# Make images with corrsponding scales
op <- par(no.readonly = TRUE)
layout(matrix(c(1,2,3,0,4,0), nrow=2, ncol=3), widths=c(4,4,1), heights=c(4,1))
#layout.show(4)
#1st image
breaks <- seq(min(volcano), max(volcano),length.out=100)
par(mar=c(1,1,1,1))
image(seq(dim(volcano)[1]), seq(dim(volcano)[2]), volcano,
col=pal.1(length(breaks)-1), breaks=breaks-1e-8, xaxt="n", yaxt="n", ylab="", xlab="")
#Add additional graphics
levs <- pretty(range(volcano), 5)
contour(seq(dim(volcano)[1]), seq(dim(volcano)[2]), volcano, levels=levs, add=TRUE)
#Add scale
par(mar=c(3,1,1,1))
imageScale(volcano, col=pal.1(length(breaks)-1), breaks=breaks-1e-8,axis.pos=1)
abline(v=levs)
box()

#2nd image
breaks <- c(0,100, 150, 170, 190, 200)
par(mar=c(1,1,1,1))
image(seq(dim(volcano)[1]), seq(dim(volcano)[2]), volcano,
col=pal.2(length(breaks)-1), breaks=breaks-1e-8, xaxt="n", yaxt="n", ylab="", xlab="")
#Add additional graphics
levs=breaks
contour(seq(dim(volcano)[1]), seq(dim(volcano)[2]), volcano, levels=levs, add=TRUE)
#Add scale
par(mar=c(1,1,1,3))
imageScale(volcano, col=pal.2(length(breaks)-1), breaks=breaks-1e-8,
axis.pos=4, add.axis=FALSE)
axis(4,at=breaks, las=2)
box()
abline(h=levs)
par(op)
```

`jetPal` *jet palette*

Description

`jetPal` is a palette of colors similar that found in Matlab. This is a nice general palette for data exploration - similar to a rainbow palette but does not cycle back to the lower color level (dark blue –> cyan –> yellow –> dark red).

Usage

```
jetPal(n)
```

Arguments

n	Number of colors to generate

Examples

```
image(volcano, col=jetPal(50))
```

---

lonLatFilter *Filter lon/lat positions that fall within defined boundaries*

Description

lonLatFilter Tests whether lon/lat positions fall within a defined box of lon/lat borders (location on border returns TRUE

Usage

```
lonLatFilter(lon_vector, lat_vector, west, east, south, north)
```

Arguments

lon_vector	Longitude 1 (in decimal degrees)
lat_vector	Latitude 1 (in decimal degrees)
west	West longitude border (decimal degrees)
east	East longitude border (decimal degrees)
south	South latitude border (decimal degrees)
north	North latitude border (decimal degrees)

Value

Vector of position inclusion in the defined lon/lat borders

Examples

```
set.seed(1)
n <- 1000
Pos <- list(lon=runif(n, min=-180, max=180), lat=runif(n, min=-180, max=180))
# Check to see if positions are withing boundaries
res <- lonLatFilter(Pos$lon, Pos$lat, -20, 20, -40, 40)
# Plot
op <- par(mar=c(4,4,1,1))
plot(Pos$lon, Pos$lat, pch=21, col=1, bg=2*res)
rect(-20, -40, 20, 40, border=3)
par(op)
```

matrixPoly *Make polygons from a matrix*

Description

matrixPoly creates a list of polygon coordinates given a matrix z and corresponding x and y coordinates for the dimensions of z.

Usage

```
matrixPoly(x, y, z, n = NULL)
```

Arguments

x,y	Optional vectors of values for matrix z rows (x) and columns (y). If excluded, the function assumes the values to be a evenly-spaced sequence from 0 to 1.
z	A matrix
n	An optional vector of element positions of z for polygon creation

Value

List of polygon coordinates for each element of z

Examples

```
# Make sythetic data
set.seed(1)
m=8
n=10
x <- seq(m)
y <- seq(n)
z <- matrix(runif(m*n), nrow=m, ncol=n)

# Ex 1 - add another image layer
image(x, y, z, col=grey.colors(20))
N <- sample(1:(m*n),20)
z2 <- NaN*z
z2[N] <- 1
image(x, y, z2, col=rgb(0,0,1,0.4), add=TRUE)
box()

# Ex 2 - add polygons
image(x, y, z, col=grey.colors(20))
poly <- matrixPoly(x, y, z=z, n=N)
sapply(poly, function(X){polygon(X, col=rgb(1,1,0,0.4), border=1)})
box()

# Ex 3 - add polygons to unequal grid
x2 <- cumsum(round(runif(m, min=1, max=10)))
y2 <- cumsum(round(runif(n, min=1, max=10)))
image(x2, y2, z, col=grey.colors(20))
poly <- matrixPoly(x2, y2, z=z, n=N)
sapply(poly, function(X){polygon(X, col=rgb(1,0,0,0.4), border=1)})
box()
```

nearest — *Calculate the nearest element in a vector as compared to a reference value*

Description

the nearest function returns the position of a vector that is closest to a defined value by determining the element with the smallest squared distance.

Usage

```
nearest(value, lookup_vector)
```

Arguments

value	A numeric reference value
lookup_vector	The vector to compare to the reference value

Value

Vector element index of nearest value

Examples

```
set.seed(1)
x <- runif(10, min=0, max=100)
res <- nearest(50, x)
plot(x)
abline(h=50, col=8, lty=2)
points(res, x[res], pch=20, col=2)
```

newLonLat — *Directional bearing between two geographic locations*

Description

newLonLat calculates a new lon/lat position given an sarting lon/lat position, a bearing and a distance to the new lon/lat position. Either the lon and lat arguments or the bearing and distance arguments can be a vector, whereby a vector of new locations will be calculated.

Usage

```
newLonLat(lon, lat, bearing, distance)
```

Arguments

lon	Longitude 1 (in decimal degrees)
lat	Latitude 1 (in decimal degrees)
bearing	Longitude 2 (in decimal degrees)
distance	Distance to new lon/lat position (km)

Value

List of new lon/lat locations

Examples

```
# Single new lon/lat position calculation
newLonLat(0,0,45,1000)

# Vector of new lon/lat positions and plot
startPos <- list(lon=0,lat=0)
endPos <- newLonLat(startPos$lon, startPos$lat, seq(0,360,20), 1000)
plot(1, t="n", xlim=range(endPos$lon), ylim=range(endPos$lat), xlab="lon", ylab="lat")
segments(startPos$lon, startPos$lat, endPos$lon, endPos$lat, col=rainbow(length(endPos$lon)))
points(startPos$lon, startPos$lat)
points(endPos$lon, endPos$lat)
```

---

plotStacked *Stacked plot*

Description

plotStacked makes a stacked plot where each y series is plotted on top of each other using filled polygons.

Usage

```
plotStacked(x, y, order.method = "as.is", ylab = "", xlab = "",
  border = NULL, lwd = 1, col = rainbow(length(y[1, ])), ylim = NULL,
  ...)
```

Arguments

x	A vector of values
y	A matrix of data series (columns) corresponding to x
order.method	Method of ordering y plotting order. One of the following: c("as.is", "max", "first"). "as.is" - plot in order of y column. "max" - plot in order of when each y series reaches maximum value. "first" - plot in order of when each y series first value > 0.
ylab	y-axis labels
xlab	x-axis labels
border	Border colors for polygons corresponding to y columns (will recycle) (see ?polygon for details)
lwd	Border line width for polygons corresponding to y columns (will recycle)
col	Fill colors for polygons corresponding to y columns (will recycle).
ylim	y-axis limits. If ylim=NULL, defaults to c(0, 1.2*max(apply(y,1,sum)).
...	Other plot arguments

Examples

```
#Create data
set.seed(1)
m <- 500
n <- 30
x <- seq(m)
y <- matrix(0, nrow=m, ncol=n)
colnames(y) <- seq(n)
for(i in seq(ncol(y))){
 mu <- runif(1, min=0.25*m, max=0.75*m)
 SD <- runif(1, min=5, max=20)
 TMP <- rnorm(1000, mean=mu, sd=SD)
 HIST <- hist(TMP, breaks=c(0,x), plot=FALSE)
 fit <- smooth.spline(HIST$counts ~ HIST$mids)
 y[,i] <- fit$y
}
y <- replace(y, y<0.01, 0)

#Ex.1 : Color by max value)
pal <- colorRampPalette(c(rgb(0.85,0.85,1), rgb(0.2,0.2,0.7)))
BREAKS <- pretty(apply(y,2,max),8)
LEVS <- levels(cut(1, breaks=BREAKS))
COLS <- pal(length(BREAKS )-1)
z <- val2col(apply(y,2,max), col=COLS)

#Create stacked plot (plot order = "as.is")
plotStacked(x,y, xlim=c(100, 400), ylim=c(0, 1.2*max(apply(y,1,sum), na.rm=TRUE)),
yaxs="i", col=z, border="white", lwd=0.5)

#Create stacked plot (plot order = "max")
plotStacked(x,y, xlim=c(100, 400), ylim=c(0, 1.2*max(apply(y,1,sum), na.rm=TRUE)),
order.method="max", yaxs="i", col=z, border="white", lwd=0.5)

#Ex. 2 : Color by first value
ord <- order(apply(y, 2, function(r) min(which(r>0))))
y2 <- y[, ord]
pal <- colorRampPalette(c("blue", "cyan", "yellow", "red"))
z <- pal(ncol(y2))

#Create stacked plot (plot order = "as.is")
plotStacked(x,y2, xlim=c(100, 400), ylim=c(0, 1.2*max(apply(y2,1,sum), na.rm=TRUE)),
yaxs="i", col=z, border=1, lwd=0.25)

#Create stacked plot (plot order = "max")
plotStacked(x,y2, xlim=c(100, 400), ylim=c(0, 1.2*max(apply(y2,1,sum), na.rm=TRUE)),
order.method="max", yaxs="i", col=z, border=1, lwd=0.25)
```

`plotStream` *Stream plot*

Description

`plotStream` makes a "stream plot" where each `y` series is plotted as stacked filled polygons on alternating sides of a baseline. A random wiggle is applied through the arguments `frac.rand` and `spar` such that each plot will be different unless preceeded by a random seed (e.g. `set.seed(1)`).

Usage

```
plotStream(x, y, order.method = "as.is", frac.rand = 0.1, spar = 0.2,
  center = TRUE, ylab = "", xlab = "", border = NULL, lwd = 1,
  col = rainbow(length(y[1, ])), ylim = NULL, ...)
```

Arguments

x	A vector of values
y	A matrix of data series (columns) corresponding to x
order.method	Method of ordering y plotting order. One of the following: c("as.is", "max", "first"). "as.is" - plot in order of y column. "max" - plot in order of when each y series reaches maximum value. "first" - plot in order of when each y series first value > 0.
frac.rand	Fraction of the overall data "stream" range used to define the range of random wiggle (uniform distrubution) to be added to the baseline g0
spar	Setting for smooth.spline function to make a smoothed version of baseline "g0"
center	Logical. If TRUE, the stacked polygons will be centered so that the middle, i.e. baseline (g0), of the stream is approximately equal to zero. Centering is done before the addition of random wiggle to the baseline.
ylab	y-axis labels
xlab	x-axis labels
border	Border colors for polygons corresponding to y columns (will recycle) (see ?polygon for details)
lwd	Border line width for polygons corresponding to y columns (will recycle)
col	Fill colors for polygons corresponding to y columns (will recycle).
ylim	y-axis limits. If ylim=NULL, defaults to c(-0.7, 0.7)*max(apply(y,1,sum))
...	Other plot arguments

Value

A plot with stream visualization added

Examples

```
#Create data
set.seed(1)
m <- 500
n <- 30
x <- seq(m)
y <- matrix(0, nrow=m, ncol=n)
colnames(y) <- seq(n)
for(i in seq(ncol(y))){
 mu <- runif(1, min=0.25*m, max=0.75*m)
 SD <- runif(1, min=5, max=20)
 TMP <- rnorm(1000, mean=mu, sd=SD)
 HIST <- hist(TMP, breaks=c(0,x), plot=FALSE)
 fit <- smooth.spline(HIST$counts ~ HIST$mids)
 y[,i] <- fit$y
}
y <- replace(y, y<0.01, 0)
```

```
#Ex.1 : Color by max value)
pal <- colorRampPalette(c(rgb(0.85,0.85,1), rgb(0.2,0.2,0.7)))
BREAKS <- pretty(apply(y,2,max),8)
LEVS <- levels(cut(1, breaks=BREAKS))
COLS <- pal(length(BREAKS )-1)
z <- val2col(apply(y,2,max), col=COLS)

#Plot order = "as.is"
plotStream(x,y, xlim=c(100, 400), center=TRUE, spar=0.3, frac.rand=0.2,
col=z, border="white", lwd=0.5)

#Plot order = "max"
plotStream(x,y, xlim=c(100, 400), center=TRUE, order.method="max", spar=0.3,
frac.rand=0.2, col=z, border="white", lwd=0.5)

#Ex. 2 : Color by first value
ord <- order(apply(y, 2, function(r) min(which(r>0))))
y2 <- y[, ord]
pal <- colorRampPalette(c("blue", "cyan", "yellow", "red"))
z <- pal(ncol(y2))

#Plot order = "as.is"
plotStream(x,y2, xlim=c(100, 400), center=FALSE, spar=0.1, frac.rand=0.05,
col=z, border=1, lwd=0.25)

#Plot order = "max"
plotStream(x,y2, xlim=c(100, 400), center=FALSE, order.method="max", spar=0.1,
frac.rand=0.05, col=z, border=1, lwd=0.25)

#Extremely wiggly, no borders, no box, no axes, black background
op <- par(bg=1, mar=c(0,0,0,0))
plotStream(x,y2, xlim=c(100, 400), center=FALSE, spar=0.3, frac.rand=1, col=z,
border=NA, axes=FALSE)
par(op)
```

`round2reso` *Round to defined resolution increment*

Description

`round2res` rounds a value to a given resolution (e.g. increments of 0.5) rather than the typical decimal place

Usage

```
round2reso(val, reso)
```

Arguments

`val`	Vector. The values to be rounded
`reso`	Numeric. The resolution or increment to use for rounding

Examples

```
set.seed(1)
n <- 10
x <- runif(n, min=0, max=20)
xr <- round2reso(x, 5) # rounded values to increments of 5
plot(x, t="n", ylim=c(0,20), pch=20)
abline(h=seq(0,20,5), col=8, lty=3)
points(x, col=1, pch=1)
points(xr, col=2, pch=20)
arrows(x0=seq(n), x1=seq(n), y0=x, y1=xr, length=0.1)
```

---

spirographR *Make a sprirograph-like design*

Description

spirographR will produce sprirograph-like design as either a hypotrochoid or an epitrochoid (depending on whether radius A or B is larger).

Usage

```
spirographR(x = 0, y = 0, a.rad = 1, b.rad = -4, bc = -2, rev = 4,
  n.per.rev = 360)
```

Arguments

x,y	Center coordinates of stationary circle 'a'
a.rad	Radius of stationary circle 'a'
b.rad	Radius of circle 'b' travelling around stationary circle 'a'
bc	Distance from the center of 'b' to a point 'c' which will turn with b as if attached to a stick.
rev	Number of revolutions that 'b' should travel around 'a'
n.per.rev	Number of radial increments to be calculated per revolution

Details

A positive value for 'b' will result in a epitrochoid, while a negative value will result in a hypotrochoid.

Examples

```
op <- par(mar=c(0,0,0,0), bg=1)
plot(spirographR(), t="l", col=6, lwd=3)
plot(spirographR(a.rad=1, b.rad=3.5, rev=7), t="l", col=7, lwd=3)
plot(spirographR(a.rad=4.1, b.rad=-6, rev=100, bc=2.3), t="l", col=5, lwd=1)
par(op)
```

`val2col` *Convert values to color levels*

Description

The `val2col` function converts a vector of values("z") to a vector of color levels. One must define the number of colors. The limits of the color scale ("zlim") or the break points for the color changes("breaks") can also be defined. When breaks and zlim are defined, breaks overrides zlim. All arguments are similar to those in the `image` function.

Usage

```
val2col(z, zlim, col = heat.colors(12), breaks)
```

Arguments

`z`	A vector of values (default is 12 colors from the `heat.colors` palette).
`zlim`	Limits of the color scale values.
`col`	Vector of color values
`breaks`	Break points for color changes. If breaks is specified then `zlim` is unused and the algorithm used follows `cut`, so intervals are closed on the right and open on the left except for the lowest interval which is closed at both ends.

Examples

```
set.seed(1)
n <- 250
x <- seq(n)
y <- rnorm(n)

# Use all levels, evenly distributed breaks
Col <- val2col(y, col=rainbow(20))
plot(x,y, pch=21, bg=Col)

# Use limits, evenly distributed breaks
pal <- colorRampPalette(c("blue", "cyan", "yellow", "red"))
Col <- val2col(y, zlim=c(-1,1), col=pal(20))
plot(x,y, pch=21, bg=Col)
abline(h=c(-1,1), col=8, lty=2)

# Use custom breaks (break vector must have one more break than color)
Col <- val2col(y, col=topo.colors(6), breaks=c(-Inf, -2, -1, 0, 1, 2, Inf))
plot(x,y, pch=21, bg=Col)
abline(h=c(-Inf, -2, -1, 0, 1, 2, Inf), col=8, lty=2)
```

Index