Package 'sinkr'

February 23, 2016

Type Package

Title sinkr

Version 1.2.0

Date 2015-03-31

Author Marc Taylor <ningkiling@gmail.com>

Maintainer Marc Taylor <ningkiling@gmail.com>

Depends R (>= 2.10)

Imports irlba, rARPACK, maps, vegan, ncdf4, Matrix

Description The sinkr package contains a collection of functions with some emphasis on multivariate methods and handling of geographic data fields.

License MIT + file LICENSE

URL https://github.com/marchtaylor/sinkr

RoxygenNote 5.0.1

NeedsCompilation no

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 dissimilarity "bray".
var.dist.method	The method of calculating dissimilarity indices bewteen samples in the variable matrix. Defaults to Euclidean dissimilarity "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)
```

`colorPalette` *Color interpolation with uneven step size*

Description

Color ramp with differing number of steps between color levels. Wrapper for `colorRamp`.

Usage

```
colorPalette(steps, n.steps.between = NULL, ...)
```

Arguments

`steps`	colors to interpolate; must be a valid argument to `col2rgb()`.
`n.steps.between`	number of color steps in between each color. Allows one to strech out specified colors more than others. Defaust is that all steps have the same weighting.
`...`	arguments to pass to `colorRamp`.

Details

This is a wrapper function for colorRampPalette. It allows for the definition of the number of intermediate colors between the main colors. Using this option one can stretch out colors that should predominate the palette spectrum. Additional arguments of colorRampPalette can also be added regarding the type and bias of the subsequent interpolation..

Examples

```
# Color scales with and without steps in between
op <- par(mfcol=c(2,1), omi=c(0.1,0.1,0.1,0.1), mai=c(1,0.2,0.2,0.2))
steps <- c("blue4", "cyan", "white", "yellow", "red4")
pal <- colorPalette(steps, space="rgb")
z=1:1000
imageScale(z, col=pal(41))
box()
steps <- c("blue4", "cyan", "white", "yellow", "red4")
pal <- colorPalette(steps, c(20,1,1,20), space="rgb")
z=1:1000
imageScale(z, col=pal(41))
box()
par(op)

# Use of transparency in palette (via alpha=TRUE)
op <- par(mar=c(2,2,2,2))
snow <- replace(volcano,  volcano<150, NaN) * 1e-8*volcano^3
elevation.pal <- colorPalette(c("black", "blue", "red"), c(1,6))
snow.pal <- colorPalette(c(rgb(0.9,0.9,0.9,0), rgb(0.9,0.9,0.9,1)), alpha=TRUE)
image(volcano, col=elevation.pal(100), axes=FALSE)
image(snow, col=snow.pal(100), add=TRUE)
contour(volcano, add=TRUE, levels=150, col="white", lwd=2, cex=2)
text(0.3, 0.9, "Snow line", col="white")
par(op)
```

---

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,
  method = "svds")
```

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% of the non-gap values (which ever is larger) will be sampled at random.
delta.rms	The threshold for RMS convergence.
method	Method to use for matrix decomposition ("irlba", "svd", or "svds"). Default is method="svds", which is more computationally efficient for large matrices. Use method="svd" for small matrices where a full set of EOFs will need to be produced before delta.rms converges. method="irlba" can also be used for partial decomposition, and is included for consistency with previous versions of the sinkr package, but is now replaced by method="svds" as a default method due to better performance.

Details

Method svds is now the default as it provides better estimates of trailing EOFs than irlba and can be computationally faster during later iterations where multiple singular vectors are calculated.

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, J-M, 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)

# 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

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

# The dineof "interpolated" field
set.seed(1)
RES <- dineof(Xo, delta.rms = 1e-03) # lower 'delta.rms' for higher resolved interpolation
Xa <- RES$Xa

# Visualization 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)




### Example with iris dataset
iris2 <- as.matrix(iris[,1:4]) # only use numeric morphometric data
frac.gaps <- 0.3 # fraction NaN values

# make gappy dataset
set.seed(1)
```

```
gaps <- sample(seq(length(iris2)), frac.gaps*length(iris2))
iris2g <- replace(iris2, gaps, NaN)

# The dineof "interpolated" field
# irlba should produce warning due to large percentage of total singular values
# used in interpolation
set.seed(1)
RES <- dineof(iris2g, delta.rms = 1e-02)

# using method="svd" is better
set.seed(1)
RES <- dineof(iris2g, delta.rms = 1e-02, method="svd",
 ref.pos = RES$ref.pos
)

# plot results
plot(iris2, RES$Xa,
     col=rep(rainbow(ncol(iris2)), each=nrow(iris2)),
     pch=as.numeric(iris$Species)
)
abline(0,1,col=8, lty=1)
legend("topleft", legend=colnames(iris2), col=rainbow(ncol(iris2)), lty=1, bty = "n")
legend("bottomright", legend=levels(iris$Species), pch=1:3, bty = "n")
sqrt(mean((iris2[gaps] - RES$Xa[gaps])^2, na.rm=TRUE)) # root mean square error


# Note: The use of dineof on small matrices may result in
# an overfitted interpolation if too many reference points are used.
# Use of eof(, recursive=TRUE) may provide better estimates - i.e. "RSEOF" method
# Example:

# EOF
E <- eof(iris2g, recursive = TRUE)

# Determine number of significant EOFs
En <- eofNull(iris2g, recursive = TRUE, nperm = 99)
En$n.sig

# reconstruction with significant EOFs
R <- eofRecon(E, pcs = seq(En$n.sig))
R[-gaps] <- iris2g[-gaps] # replace non-gap values

# plot interpolated values
plot(iris2, R,
     col=rep(rainbow(ncol(iris2)), each=nrow(iris2)),
     pch=as.numeric(iris$Species)
)
abline(0,1,col=8, lty=1)
legend("topleft", legend=colnames(iris2), col=rainbow(ncol(iris2)), lty=1, bty = "n")
legend("bottomright", legend=levels(iris$Species), pch=1:3, bty = "n")
sqrt(mean((iris2[gaps] - R[gaps])^2, na.rm=TRUE)) # root mean square error
```

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`', '`svds`'). Defaults to 'svd' when `method = NULL` and 'svds' when `method = NULL` and `recursive = TRUE`. Use of 'svds' or 'irlba" is recommended when `recursive = TRUE` due to faster computation speed. All methods should give identical results when `recursive = TRUE`. When `recursive = FALSE`, `svd` and `eigen` give similar results for non-gappy fields, but will differ slightly with gappy fields due to decomposition of a non-positive 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 (i.e. `recursive = TRUE`) 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 estimate of EOFs. However, RSEOF can be much faster in cases where the number of columns in `F1` is smaller than the number of rows. Faster computation time with `svds` over `irlba` may not result when `recursive = TRUE` due to computation of leading vectors only.

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').
`tot.var`	Total variance of field `F1` (from `cov4gappy`).
`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])
SIGN <- diag(sign(diag(cor(Et$A, Pt$x)))) # correction for differing sign
matplot(Et$A %*% SIGN, Pt$x)
abline(0,1, col=8)

# 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)


### EOF of climate field example
library(maps) # required package to add map

# load data
data(sst)
names(sst)

# EOF
E <- eof(sst$field, nu=10)

# Plot of EOF loading and PC
eof.num <- 2 # EOF number to plot
op <- par(no.readonly=TRUE)
layout(matrix(c(1,3,2,3),nrow=2, ncol=2), widths=c(5,1), heights=c(3,3), respect=TRUE)
op <- par(ps=10, cex=1)
par(mar=c(4,4,1,1))
PAL <- colorPalette(c("blue", "cyan", "grey90", "yellow", "red"), c(10,1,1,10))
ZLIM <- c(-1,1)*max(abs(E$u[,eof.num]))
COL <- val2col(E$u[,eof.num], col=PAL(100), zlim=ZLIM)
plot(lat ~ lon, data=sst$grid, pch=22, bg=COL, col=COL, cex=2)
map("world", add=TRUE)
par(mar=c(4,0,1,4))
imageScale(E$u[,eof.num], col=PAL(100), zlim=ZLIM, axis.pos=4)
par(mar=c(4,4,1,4))
plot(sst$date, E$A[,eof.num], t="l", xlab="date", ylab="")
lines(loess.smooth(sst$date, E$A[,eof.num], span=1/3), col=rgb(0.5,0.5,1), lwd=2) # smoothed signal
abline(h=0, v=seq(as.Date("1000-01-01"), as.Date("2100-01-01"), by="10 years"), col=8, lty=3)
par(op)
```

---

eofBoot *Calculate number of non-mixed EOFs (eof version)*

Description

The eofBoot function uses a bootstrap randomization approach to calculate distributions of Empirical Orthogonal Function analysis (EOF) singular values with the eof function. EOF mode

significance is assessed against the distributions of neighboring EOF singular values ("Lambda") calculated by the permutated models. A bootstrap routine follows the procedure of Babamoradi et al. (2013) whereby permutations sample rows (samples) more than once, which is a non-parametric approach does not make assumptions about the distribution of data.

Usage

```
eofBoot(F1, centered = TRUE, scaled = FALSE, nu = NULL, method = NULL,
  recursive = FALSE, nperm = 99)
```

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). See eof for details
nperm	Numeric. The number of null model permutations to calculate.

References

Babamoradi, H., van den Berg, F., Rinnan, A, 2013. Bootstrap based confidence limits in principal component analysis - A case study, Chemometrics and Intelligent Laboratory Systems, Volume 120, pp. 97-105. doi:10.1016/j.chemolab.2012.10.007.

Examples

```
# Generate data
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

# True field
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="+"))

Xt <- t(Xt)

# Noise 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

# True field + Noise field
Xp <- Xt + R

res <- eofBoot(Xp, centered=FALSE, scaled=FALSE, nperm=499)
ylim <- range(res$Lambda.orig, res$Lambda)
boxplot(res$Lambda, log="y", col=8, border=2, outpch="", ylim=ylim)
points(res$Lambda.orig)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Non-mixed PCs =", res$n.sig), side=3, line=0.5, col=4)
```

`eofNull` *Calculate significance of EOFs compared to a null model (eof version)*

Description

The `eofNull` function uses a randomization approach to calculate a null model for use in Empirical Orthogonal Function analysis (EOF) with the `eof` function. EOF mode significance is assessed against the distribution of EOF singular values ("Lambda") calculated by the null models

Usage

```
eofNull(F1, centered = TRUE, scaled = FALSE, nu = NULL, method = NULL,
  recursive = FALSE, nperm = 99)
```

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). See eof for details
nperm	Numeric. The number of null model permutations to calculate.

Examples

```
# Generate data
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

# True field
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="+"))

Xt <- t(Xt)

# Noise 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

# True field + Noise field
Xp <- Xt + R

res <- eofNull(Xp, method="svd", centered=FALSE, scaled=FALSE, nperm=499)
ylim <- range(res$Lambda.orig, res$Lambda)
boxplot(res$Lambda, log="y", col=8, border=2, outpch="", ylim=ylim)
points(res$Lambda.orig)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Significant PCs =", res$n.sig), side=3, line=0.5, col=4)
```

eofPred *EOF prediction*

Description

The eofPred function predics the principal component loadings of a new dataset with an eof object

Usage

```
eofPred(EOF, newdata = NULL, pcs = NULL)
```

Arguments

EOF	An object resulting from the function eof
newdata	new data to project onto eofs
pcs	The principal components (PCs) to use in the reconstruction (defaults to the full set of PCs: pcs=seq(ncol(EOF$u)))

Value

A matrix of principal component loadings (as in EOF$A)

Examples

```
### Non-gappy example
# Data
set.seed(1)
ptrain <- 0.5 # portion of data to use for training set
tmp <- sample(nrow(iris), nrow(iris)*ptrain)
train <- iris[tmp,1:4] # training set
valid <- iris[-tmp,1:4] # validation set

# EOF analysis
Efull <- eof(iris[,1:4], centered=TRUE, scaled=TRUE) # EOF of full data
Etrain <- eof(train, centered=TRUE, scaled=TRUE) # EOF of training data

# Null model
Efull.n.sig <- eofNull(iris[,1:4], centered=TRUE, scaled=TRUE, nperm = 99)$n.sig
Etrain.n.sig <- eofNull(train, centered=TRUE, scaled=TRUE, nperm = 99)$n.sig

# Predict PCs of validation set
pred <- eofPred(Etrain, newdata=valid)
# plot against full data-derived PCs
SIGN <- diag(sign(diag(cor(Efull$A[-tmp,], pred)))) # correction for differing sign
matplot(Efull$A[-tmp,] %*% SIGN, pred)
abline(0,1, col=8)
legend("topleft", legend=paste("PC", seq(ncol(pred))),
       col=seq(ncol(pred)), lty=0, pch=1)

# Predict PCs of full set
pred <- eofPred(Efull, newdata=iris[,1:4])
# plot against full data-derived PCs (should be equal)
SIGN <- diag(sign(diag(cor(Efull$A, pred)))) # correction for differing sign
```

```
matplot(Efull$A %*% SIGN, pred)
abline(0,1, col=8)
legend("topleft", legend=paste("PC", seq(ncol(pred))),
       col=seq(ncol(pred)), lty=0, pch=1)


### gappy example
# Data
set.seed(1) #
ptrain <- 0.5 # portion of data to use for training set
pgap <- 0.2 # portion of data that are gaps (e.g. NaNs)
irisg <- as.matrix(iris[,1:4])
irisg[sample(length(irisg), length(irisg)*pgap)] <- NaN
tmp <- sample(nrow(irisg), nrow(irisg)*ptrain)
traing <- irisg[tmp,] # training set
validg <- irisg[-tmp,] # validation set

# EOF analysis
Efullg <- eof(irisg, centered=TRUE, scaled=TRUE, recursive=TRUE) # EOF of full data
Etraing <- eof(traing, centered=TRUE, scaled=TRUE, recursive=TRUE) # EOF of training data

# EOF Null model
EfullgNull <- eofNull(irisg, centered=TRUE, scaled=TRUE, recursive=TRUE, nperm=99)
EtraingNull <- eofNull(traing, centered=TRUE, scaled=TRUE, recursive=TRUE, nperm=99)

# Predict PCs of validation set
pred <- eofPred(Etrain, newdata=validg)
# plot against full data-derived PCs
SIGN <- diag(sign(diag(cor(Efullg$A[-tmp,], pred)))) # correction for differing sign
matplot(Efullg$A[-tmp,] %*% SIGN, pred)
abline(0,1, col=8)
legend("topleft", legend=paste("PC", seq(ncol(pred))),
       col=seq(ncol(pred)), lty=0, pch=1)

# Reconstruction and measurement of error against non-gappy data
usePCs <- seq(1) # Efull.n.sig = 1
Rg <- eofRecon(Etrain, pcs=seq(usePCs), newpcs=pred)
plot( c(as.matrix(iris[-tmp,1:4])), c(Rg) )
abline(0,1, col=8)
rmse <- sqrt( mean( ( c(as.matrix(iris[-tmp,1:4])) - c(Rg)   )^2, na.rm=TRUE) )
rmse
```

---

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, newpcs = NULL)
```

Arguments

EOF	An object resulting from the function eof.
pcs	The principal components (PCs) to use in the reconstruction (defaults to the full set of PCs: pcs=seq(ncol(EOF$u)))
newpcs	A matrix of new principal coordinate to use in the reconstruction. This would typically come from a gappy dataset whose missing values are to be predicted based on the EOF loadings of a the EOF object (see).

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) )
```

`fieldAnomaly` *Calculate the daily (julian or month/day date) or monthly anomaly of a data field*

Description

The `fieldAnomaly` function calculates an anomaly field by subtracting the daily or monthly means from each spatial location (columns) of a field.

Usage

```
fieldAnomaly(y, x, level = "month")
```

Arguments

`y`	A matrix of the spatio-temporal field with rows being the temporal dimension and columns being the spatial dimension
`x`	A vector of the class "Date", "POSIXct", or "POSIXlt" containing the dates that correspond to the rows of matrix `y`.
`level`	Character string. Anomaly to compute (`"julian"`, `"day"` or `"month"`). `"julian"` will consider unique `as.POSIXlt(x)$yday` values, while `"day"` will consider unique `format(x,"%m%d")` strings.

Examples

```
set.seed(1)
Time <- seq.Date(as.Date("1990-01-01"), as.Date("1996-12-31"), by="day")
Space <- seq(pi,2*pi,,10)
Signal1 <- sin(as.POSIXlt(Time)$yday/365*(2*pi)) # annual signal
Signal2 <- sin(as.POSIXlt(Time)$yday/365*(24*pi)) # monthly signal
cumSignal <- 1*Signal1+0.3*Signal2
plot(Time, cumSignal, t="l")
abline(v=seq(as.Date("1900-01-01"), as.Date("2100-01-01"), by="year"), lty=2, col=8)
Z <- outer(cumSignal, sin(Space))
Noise <- Z * 0 * array(rnorm(length(Z)), dim=dim(Z))
Z <- Z + Noise

# anomaly by month (removes "Signal1")
Z.anom <- fieldAnomaly(y=Z, x=Time, level="month")
zran <- c(-1,1) * max(abs(range(Z, Z.anom)))
pal <- colorPalette(c("blue4", "cyan", "grey90", "yellow", "red4"), c(20,1,1,20))
op <- par(no.readonly=TRUE)
layout(matrix(c(1,2,3,3), nrow=2, ncol=2), widths=c(4,1), heights=c(2,2))
```

```
par(mar=c(5,5,3,1))
image(Time, Space, Z, col=pal(101), zlim=zran, main="Original")
image(Time, Space, Z.anom, col=pal(101), zlim=zran, main="Anomaly")
par(mar=c(5,0,3,5))
imageScale(Z, col=pal(101), zlim=zran, axis.pos=4)
mtext("Value", side=4, line=3)
par(op)


# anomaly by day of year (i.e. date, removes "Signal1" and "Signal2")
Z.anom <- fieldAnomaly(y=Z, x=Time, level="day")
zran <- c(-1,1) * max(abs(range(Z, Z.anom)))
pal <- colorPalette(c("blue4", "cyan", "grey90", "yellow", "red4"), c(20,1,1,20))
op <- par(no.readonly=TRUE)
layout(matrix(c(1,2,3,3), nrow=2, ncol=2), widths=c(4,1), heights=c(2,2))
par(mar=c(5,5,3,1))
image(Time, Space, Z, col=pal(101), zlim=zran, main="Original")
image(Time, Space, Z.anom, col=pal(101), zlim=zran, main="Anomaly")
par(mar=c(5,0,3,5))
imageScale(Z, col=pal(101), zlim=zran, axis.pos=4)
mtext("Value", side=4, line=3)
par(op)


# anomaly by julian day (day of year, removes "Signal1" and "Signal2")
Z.anom <- fieldAnomaly(y=Z, x=Time, level="julian")
zran <- c(-1,1) * max(abs(range(Z, Z.anom)))
pal <- colorPalette(c("blue4", "cyan", "grey90", "yellow", "red4"), c(20,1,1,20))
op <- par(no.readonly=TRUE)
layout(matrix(c(1,2,3,3), nrow=2, ncol=2), widths=c(4,1), heights=c(2,2))
par(mar=c(5,5,3,1))
image(Time, Space, Z, col=pal(101), zlim=zran, main="Original")
image(Time, Space, Z.anom, col=pal(101), zlim=zran, main="Anomaly")
par(mar=c(5,0,3,5))
imageScale(Z, col=pal(101), zlim=zran, axis.pos=4)
mtext("Value", side=4, line=3)
par(op)


# test that anomalies are removed correctly
Zalt <- as.matrix(apply(scale(Z),1,mean))
plot(Time, Zalt, t="l")

tmp <- aggregate(Zalt ~ as.POSIXlt(Time)$yday, FUN=mean)
plot(strptime(paste("2000",tmp[,1], sep="-"), format="%Y-%j"), tmp[,2],
     t="o", xlab="", ylab="n values", main="level = 'julian'")

tmp <- aggregate(Zalt ~ format(Time, "%m-%d"), FUN=mean)
plot(strptime(paste("2000",tmp[,1], sep="-"), format="%Y-%m-%d"), tmp[,2],
     t="o", xlab="", ylab="n values", main="level = 'day'")


Z.anom <- fieldAnomaly(y=Zalt, x=Time, level="month")
sqrt(mean(Z.anom^2))
plot(Time, Z.anom, t="l")

Z.anom <- fieldAnomaly(y=Zalt, x=Time, level="day")
```

```
sqrt(mean(Z.anom^2))
plot(Time, Z.anom, t="l")

Z.anom <- fieldAnomaly(y=Zalt, x=Time, level="julian")
sqrt(mean(Z.anom^2))
plot(Time, Z.anom, t="l")
```

`getcolors` *Select colors visually*

Description

The `getcolors` function allows one to select from a visual palette of 216 colors (i.e. color combinations from 6 levels of red, green, and blue).

Usage

```
getcolors(n)
```

Arguments

`n`	Numeric value of the number of colors to select

Value

A vector of hexadecimal codings for colors (as in `rgb`).

Examples

```
# Make synthetic data
set.seed(1)
n <- 100
x <- seq(n)
y1 <- cumsum(rnorm(n))
y2 <- cumsum(rnorm(n))
y3 <- cumsum(rnorm(n))
y4 <- cumsum(rnorm(n))
ylim <- range(c(y1,y2,y3,y4))

# Select colors
COLS <- getcolors(4)

# Plot data with selected colors
plot(x, y1, ylim=ylim, t="l", col=COLS[1], lwd=3, ylab="")
lines(x, y2, col=COLS[2], lwd=3)
lines(x, y3, col=COLS[3], lwd=3)
lines(x, y4, col=COLS[4], lwd=3)
legend("topleft", legend=paste("y", 1:4, sep=""), col=COLS, lwd=3)
```

getcolors2 *Select colors visually*

Description

The getcolors2 function allows one to select from a visual palette of 1200 colors (i.e. color combinations from 6 levels of red, green, and blue).

Usage

```
getcolors2(n)
```

Arguments

n	Numeric value of the number of colors to select

Value

A vector of hexadecimal codings for colors (as in rgb).

Examples

```
# Make synthetic data
set.seed(1)
n <- 100
x <- seq(n)
y1 <- cumsum(rnorm(n))
y2 <- cumsum(rnorm(n))
y3 <- cumsum(rnorm(n))
y4 <- cumsum(rnorm(n))
ylim <- range(c(y1,y2,y3,y4))

# Select colors
COLS <- getcolors2(4)

# Plot data with selected colors
plot(x, y1, ylim=ylim, t="l", col=COLS[1], lwd=3, ylab="")
lines(x, y2, col=COLS[2], lwd=3)
lines(x, y3, col=COLS[3], lwd=3)
lines(x, y4, col=COLS[4], lwd=3)
legend("topleft", legend=paste("y", 1:4, sep=""), col=COLS, lwd=3)
```

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 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)
```

gm_mean *Geometric Mean*

Description

Calculates the geometric mean of a vector

Usage

```
gm_mean(x, na.rm = TRUE, zero.propagate = FALSE)
```

Arguments

x	numeric vector of positive numbers.
na.rm	Remove NAs befor calculation (as in mean)
zero.propagate	Logical. Should zeros be considered (resulting in output of zero)

References

From stackoverflow answer posted by Paul McMurdie

Examples

```
### simple usage
gm_mean(c(1:4))
gm_mean(c(-1:4)) # negative values not allowed
gm_mean(c(0:4)) # zeros do not propagate
gm_mean(c(0:4), zero.propagate=TRUE) #zeros allowed to propagate
gm_mean(c(1,2,3,4, NaN)) # na.rm=TRUE
gm_mean(c(1,2,3,4, NaN), na.rm=FALSE) # na.rm=FALSE

### example of proportional change
df <- data.frame(index1 = 5, index2 = 25) # two indices of differing magnitude
mult <- c(1.25, 1.5) # multiplier
df <- rbind(df, df*mult) # indices change by differing proportions
df # view dataframe
gm_mean(mult) # mean proportional increase
gm_mean(df[2,]) / gm_mean(df[1,]) # equal
gm_mean(df[2,] / df[1,]) # equal
```

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)
```

`isin.convert` *Calculate grid and polygon coordinates for 4 km ISIN grid*

Description

This function is used to convert information regarding the ISIN grid information used by Globcolour as well as to construct associated polygons for use in mapping. The raw Globcolour .nc files come with column and row pointers as to the the grid's location. For 4.63 km resolution data ("L3b"), this translates to 4320 latitudinal rows with varying number of associated longitudinal columns depending on the latitudinal circumference. Input must be either a vector of grid numbers, `grd`, or a dataframe with column and row identifiers, `coord`. See Globcolour's "Product User Guide" for fulther details of the ISIN grid (http://www.globcolour.info/products_description.html).

Usage

```
isin.convert(grd = NULL, coord = NULL, polygons = FALSE)
```

Arguments

`grd`	numeric vector containing ISIN grid numbers needed for calculating lon/lat coordinates or polygons.
`coord`	data.frame containing ISIN grid coordinates (row and column locations). Names of columns should be `coord$row` and `coord$col`.
`polygons`	Logical. Should grid polygon corner coordinates be returned.

Value

If "`polygons = TRUE`", then returns grid information as a dataframe, including lon / lat values of grid centers. If "`polygons = TRUE`", then returns a list with polygon corners in a dataframe(longitudinal coordinates of corners (e.g. `[[i]]$x`) and latitudinal coordinates of corners (`[[i]]$y`))

`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)
```

lsos *List top n objects by size*

Description

The function lsos lists the top n objects in memory.

Usage

```
lsos(pos = 1, pattern, order.by = "Size", decreasing = TRUE,
  head = TRUE, n = 10)
```

Arguments

pos	where to look for the object (see get, 'Details'); if omitted search as if the name of the object appeared unquoted in an expression.
pattern	an optional regular expression (see ls). Only names matching pattern are returned. see glob2rx can be used to convert wildcard patterns to regular expressions.
order.by	Order list by one of the following: "Type", "Size", "Rows", "Columns"
decreasing	Logical. Decreasing in size (i.e. largest objects at top of list.) (Default=TRUE)
head	Logical (Default=TRUE). Should only the head be returned in the results of a data.frame. Argument n defines how many objects to include.
n	Numeric value (Default=10). Defines how many objects to include in results

Value

a data.frame object

References

http://stackoverflow.com/questions/1358003/tricks-to-manage-the-available-memory-in-an-r-sessio

Examples

```
x1 <- matrix(rnorm(10000), 100, 100)
x2 <- as.data.frame(matrix(rnorm(1000), 100, 10))
x3 <- rnorm(1000)
lsos()
rm(list=c("x1", "x2", "x3"))
```

makeGlobcolourField *Make a field from Globcolour data*

Description

Make a field from Globcolour data

Usage

```
makeGlobcolourField(dataPath = getwd(), yearRan = c(1997, 2013),
  lonRan = c(-83, -79), latRan = c(-16, -4), greps = c(".nc", "GSM",
  "MO"))
```

Arguments

dataPath	Path to data
yearRan	Range of years to use for field
lonRan	Range of longitude to use for field
latRan	Range of latitude to use for field
greps	Text patterns to use to filter data (i.e. filter file names that only contain these text strings)

Value

Matrix field where rows and columns are time and space, respectively

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)
```

newRange *Define a new range for a numeric*

Description

The newRange function scales a numeric vector to a new range as defined by minimum and maximum extreme values.

Usage

```
newRange(x, new.min = 0, new.max = 1)
```

Arguments

x	a numeric vector
new.min	numeric value of new minimum range value
new.max	numeric value of new maximum range value

Examples

```
y <- runif(100, min=0, max=2)
y2 <- newRange(y, new.min=0, new.max=1)
plot(y, t="l")
lines(y2, col=2)

plot(y, y2)
```

northTest *North's Rule of Thumb for EOF significance*

Description

The northTest function assesses the uniqueness of EOF modes through assumptions of error on singular values ("Lambda") as described by North et al (1982). Overlapping error limits between neighboring Lambda values indicates a possible mixure of signals. A similar test via bootstrapping method can be done with the prcompBoot or eofBoot functions.

Usage

```
northTest(x, Lambda)
```

Arguments

x	Matrix. Field used during EOF decomposition
Lambda	Vector of singular values representing variance magnitudes explained by each EOF mode. Results of prcomp will need to be squared from their standard deviation values (e.g. Lambda=res$sdev^2).

References

G.R. North, T.L. Bell, R.F. Cahalan, and F.J. Moeng. (1982). Sampling errors in the estimation of empirical orthogonal functions. Mon. Wea. Rev., 110:699-706.

Examples

```
# Generate data
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

# True field
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="+"))

Xt <- t(Xt)

# Noise 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

# True field + Noise field
Xp <- Xt + R

# eof + northTest
E <- eof(Xp, centered=FALSE, scaled=FALSE)
L <- E$Lambda # Lambdas
res <- northTest(Xp, L)
plot(L, ylim=range(c(L+res$upper.lim, L-res$lower.lim)), log="y")
segments(seq(L), L-res$lower.lim, seq(L), L+res$upper.lim, col=2)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Non-mixed PCs =", res$n.sig), side=3, line=0.5, col=4)

# prcomp + northTest
E <- prcomp(Xp, center=FALSE, scale=FALSE)
L <- E$sdev^2 # Lambdas
res <- northTest(Xp, L)
plot(L, ylim=range(c(L+res$upper.lim, L-res$lower.lim)), log="y")
segments(seq(L), L-res$lower.lim, seq(L), L+res$upper.lim, col=2)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Non-mixed PCs =", res$n.sig), side=3, line=0.5, col=4)
```

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)
```

`prcompBoot` *Calculate number of non-mixed EOFs (prcomp version)*

Description

The `prcompBoot` function uses a bootstrap randomization approach to calculate distributions of Empirical Orthogonal Function analysis (EOF) singular values with the `prcomp` function. EOF mode significance is assessed against the distributions of neighboring EOF singular values ("Lambda") calculated by the permutated models. A bootstrap routine follows the procedure of Babamoradi et al. (2013) whereby permutations sample rows (samples) more than once, which is a non-parametric approach does not make assumptions about the distribution of data.

Usage

```
prcompBoot(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL,
  nperm = 99)
```

Arguments

`x, retx, center, scale., tol`	See `prcomp` for argument definitions.
`nperm`	Numeric. The number of null model permutations to calculate.

References

Babamoradi, H., van den Berg, F., Rinnan, A, 2013. Bootstrap based confidence limits in principal component analysis - A case study, Chemometrics and Intelligent Laboratory Systems, Volume 120, pp. 97-105. doi:10.1016/j.chemolab.2012.10.007.

Examples

```
# Generate data
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

# True field
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="+"))

Xt <- t(Xt)

# Noise 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

# True field + Noise field
Xp <- Xt + R

res <- prcompBoot(Xp, center=FALSE, scale=FALSE, nperm=499)
ylim <- range(res$Lambda.orig, res$Lambda)
boxplot(res$Lambda, log="y", col=8, border=2, outpch="", ylim=ylim)
points(res$Lambda.orig)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Non-mixed PCs =", res$n.sig), side=3, line=0.5, col=4)
```

`prcompNull` *Calculate significance of EOFs compared to a null model (prcomp version)*

Description

The `prcompNull` function uses a randomization approach to calculate a null model for use in Empirical Orthogonal Function analysis (EOF) with the `prcomp` function. EOF mode significance is assessed against the distribution of EOF singular values ("Lambda") calculated by the null models

Usage

```
prcompNull(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL,
  nperm = 99)
```

Arguments

x, retx, center, scale., tol	See prcomp for argument definitions.
nperm	Numeric. The number of null model permutations to calculate.

Examples

```
# Generate data
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

# True field
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="+"))

Xt <- t(Xt)

# Noise 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

# True field + Noise field
Xp <- Xt + R

res <- prcompNull(Xp, center=FALSE, scale=FALSE, nperm=499)
ylim <- range(res$Lambda.orig, res$Lambda)
boxplot(res$Lambda, log="y", col=8, border=2, outpch="", ylim=ylim)
points(res$Lambda.orig)
abline(v=res$n.sig+0.5, lty=2, col=4)
mtext(paste("Significant PCs =", res$n.sig), side=3, line=0.5, col=4)
```

---

prcompRecon *prcomp object reconstruction*

Description

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

Usage

```
prcompRecon(pca, pcs = NULL)
```

Arguments

pca	An object resulting from the function prcomp.
pcs	The principal components ("PCs") to use in the reconstruction (defaults to the full set of PCs: pcs=seq(pca$sdev)))

Examples

```
# prcomp
P <- prcomp(iris[,1:4])

# Full reconstruction
R <- prcompRecon(P)
plot(as.matrix(iris[,1:4]), R, xlab="original data", ylab="reconstructed data")
abline(0, 1, col=2)

# Partial reconstruction
RMSE <- NaN*seq(P$sdev)
for(i in seq(RMSE)){
  Ri <- prcompRecon(P, pcs=seq(i))
  RMSE[i] <- sqrt(mean((as.matrix(iris[,1:4]) - Ri)^2))
}
plot(RMSE, t="o", xlab="Number of pcs")
abline(h=0, lty=2)
```

---

ptlocator *Data point locator*

Description

the ptlocator function allows for the selection of data points in an open graphical device via the locator function. Closest data points to the selected positions are determined via Euclidean distance following a scaling of x and y axes in order to give them equal weighting and remove the influence of differing units or ranges. Colored points are filled in for the data point that has the lowest distance to the clicked location, and the result gives the vector positions of the closest x, y data points.

Usage

```
ptlocator(n = 1, x, y, col = rgb(1, 0, 0, 0.5), pch = 20, ...)
```

Arguments

n	Number of points to select
x	Vector of x-axis values for the plotted data
y	Vector of y-axis values for the plotted data
col	Colors to use for plotting closest data points
pch	pch values to use for plotting closest data points
...	additional parameters for plotting closest data points

Value

A vector of point indices

Examples

```
set.seed(1)
n <- 200
x <- sort(runif(n, min=0, max=10*pi))
y <- sin(x) + rnorm(n, sd=0.2)

# Select 10 points at maxima and minima
plot(x, y)
pos <- ptlocator(10, x, y)
pos
```

`reltext` *Add text using relative x,y plot coordinates*

Description

`reltext` adds text using relative x,y plot coordinates. Wrapper for `text`.

Usage

```
reltext(relx = 0.5, rely = 0.5, labels = seq_along(relx), ...)
```

Arguments

`relx, rely`	numeric vectors of coordinates where the text labels should be written.
`labels`	label argument passed to `text`
`...`	other parameters passed to `text`

Examples

```
# Make data
set.seed(1)
n <- 100
a <- 3
b <- 5
x <- rnorm(n)
y <- a + b*x + rnorm(n, sd=2)
fit <- lm(y ~ x)

# Plot with regression results in top left corner
op <- par(mar=c(3,3,0.5,0.5), mgp=c(1.5,0.5,0), tcl=-0.25, las=1, bty="l")
plot(x, y)
abline(fit, lty=2, col=4)
reltext(0, 0.95, pos=4, col=4,
   labels=bquote( italic(y) == .(sprintf("%0.2f", (coef(fit)[1])))+
     .(sprintf("%0.2f", (coef(fit)[2])))~"*"~italic(x)
   )
)
```

```
reltext(0, 0.85, pos=4, col=4,
        labels=bquote( bolditalic(R)^2 == .(round(summary(fit)$adj.r.squared,2)) )
)
par(op)

# Plot multiple labels at once
plot(x, y)
abline(fit, lty=2, col=8)
reltext(c(0,0.95), c(0.95, 0.05), pos=c(4,3), col=c(3,4), labels=c("text1", "text2"))
```

---

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)
```

---

seqRan *Create a sequence of defined length over range of a vector*

Description

Create a sequence of defined length over range of a vector

Usage

```
seqRan(x, length.out = NULL, rel.ext = c(0, 0), ...)
```

Arguments

x	Numeric vector
length.out	Length of sequence to create, Defaults to length of x.
rel.ext	Relative extention of sequence limits. Default rel.xt = c(0,0) maintains original range of x
...	Additional parameters to pass to seq

Value

Vector

Examples

```
x <- c(0,1,3,4)
seqRan(x) # default settings. Returns vector of length of x
seqRan(x, 10) # using original range of x
seqRan(x, 10, c(-0.1, 0.1)) # using extended range of x
```

---

slp *Hadley SLP monthly mean dataset*

Description

The slp data set contains monthly sea level pressure data for the Equatorial Pacific within the lat/lon range 180 W - 70 W and 30 S - 30 N, and spanning the years 1971-1998.

- grid. Dateframe of lon/lat coordinates corresponding to columns of slp$field (5 deg resolution)
- date. Vector of monthly date values corresponding to rows of slp$field
- field. Matrix of sea level pressure values by month and lon/lat position.

Usage

```
data(slp)
```

Format

A list consisting of: 1. a dataframe for lon/lat positions, 2. a vector of monthly date values, and 3. a matrix of slp values by month and lon/lat position (1536 rows, 231 columns)

Source

http://www.esrl.noaa.gov/psd/gcos_wgsp/Gridded/data.hadslp2.html

Examples

```
### Ex 1. Plot of single month
library(maps)
data(slp)
plot(slp$grid, col=val2col(slp$field[1000,]), pch=".", cex=20)
map("world", add=TRUE)
```

---

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)
```

sqbin *Square binning of xy data*

Description

The sqbin function calculates frequencies in xy bins

Usage

```
sqbin(x, y, xint = NULL, yint = NULL, nxbin = 50, nybin = 50)
```

Arguments

x	a vector of x values
y	a vector of y values
xint	a vector of x intervals (end values). Defaults to values as specified by pretty(x, n=nxbin)
yint	a vector of y intervals (end values). Defaults to values as specified by pretty(y, n=nybin)
nxbin	number of x bins. Default=50. Not used when xint is defined.
nybin	number of y bins. Default=50. Not used when yint is defined.

Value

A matrix containing interval mid points (x,y), bin frequencies (z), and intervals (xint, yint).

Examples

```
# Synthetic data
set.seed(1)
n <- 1e6
x <- runif(n, min=-3, max=3)
y <- 4*x^2 + rnorm(n, sd=5)

sqbin.res <- sqbin(x,y)

# Plot
op <- par(mar=c(4,4,1,1))
image(sqbin.res, col=jetPal(20))
par(op)

# Plot with legend
op <- par(no.readonly = TRUE)
lo <- matrix(1:2, nrow=1, ncol=2)
layout(lo, widths=c(4,1), heights=c(4), respect=TRUE)
par(cex=1)
par(mar=c(3,3,1,1))
image(sqbin.res, col=jetPal(20))
par(mar=c(3,0,1,3))
imageScale(sqbin.res$z, col=jetPal(20), axis.pos=4)
par(op)
```

sst *Kaplan monthly mean anomaly dataset*

Description

The sst data set contains monthly sea surface temperature anomaly data for the Equatorial Pacific within the lat/lon range 180 W - 70 W and 30 S - 30 N, and spanning the years 1956-2014.

- grid. Dateframe of lon/lat coordinates corresponding to columns of sst$field (5 deg resolution)
- date. Vector of monthly date values corresponding to rows of sst$field
- field. Matrix of sea level pressure values by month and lon/lat position.

Usage

```
data(sst)
```

Format

A list consisting of: 1. a dataframe for lon/lat positions, 2. a vector of monthly date values, and 3. a matrix of sst anomaly values by month and lon/lat position (1906 rows, 264 columns)

Source

http://www.esrl.noaa.gov/psd/data/gridded/data.kaplan_sst.html

Examples

```
### Ex 1. Plot of single month
library(maps)
data(sst)
plot(sst$grid, col=val2col(sst$field[1000,]), pch=".", cex=20)
map("world", add=TRUE)
```

unscale *Unscale a matrix*

Description

The unscale function unscales a numeric marteix that has been either centered or scaled by the scale function. This is done by reversing the first unscaling and then uncentering based on the object's attributes.

Usage

```
unscale(x, unscale = TRUE, uncenter = TRUE)
```

Arguments

x	a numeric matrix(like object) that has been centered and/or scaled by the scale function.
unscale	a logical value defining whether to unscale x.
uncenter	a logical value defining whether to uncenter x.

Examples

```
x <- matrix(1:16, 4, 4)
xcs <- scale(x, center=TRUE, scale=TRUE) # centered and scaled
xc <- scale(x, center=TRUE, scale=FALSE) # centered only
xs <- scale(x, center=FALSE, scale=TRUE) # scaled only

# compare difference to original
x - unscale(xcs)
x - unscale(xc)
x - unscale(xs)
```

---

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