Package 'sinkr'

August 28, 2014

Type Package

Title sinkr

Version 1.0

Date 2014-07-21

Author Marc Taylor <ningkiling@gmail.com>

Maintainer Marc Taylor <ningkiling@gmail.com>

Depends irlba, png

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

License MIT

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)
```

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 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)
```

`eof` *EOF (Empirical Orthogonal Functions analysis)*

Description

This function conducts an Empirical Orthogonal Function analysis (EOF) via a covariance matrix (`cov4gappy()`) 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 cumputation 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]). 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), which is a modification of "Least Squares Empirical Orthogonal Functions" (LSEOF) (Taylor et al., 2013).

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.

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, n_pc = NULL)
```

Arguments

`EOF`	An object resulting from the function `eof`.
`n_pc`	The number of principal components (PCs) to use in the reconstruction (defaults to the fullset of PCs)

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.

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) )
```

`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)
```

---

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

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)
```

`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