Package ‘plotrix’

October 26, 2009

Version 2.7-2

Date 2009-10-25

Title Various plotting functions

Author Jim Lemon,Ben Bolker,Sander Oom, Eduardo Klein,Barry Rowlingson, Hadley Wickham,Anupam Tyagi, Olivier Eterradossi,Gabor Grothendieck, Michael Toews,John Kane,Mike Cheetham, Rolf Turner,Carl Witthoft,Julian Stander, Thomas Petzoldt,Remko Duursma,Elisa Biancotto, Ofir Levy

Maintainer Jim Lemon <jim@bitwrit.com.au>

Description Lots of plots, various labeling, axis and color scaling functions.

License GPL (>= 2)

Repository CRAN

Date/Publication 2009-10-26 19:28:20

R topics documented:

`ablineclip` *Add a straight line to a plot*

Description

As '`abline`', but has arguments '`x1,x2,y1,y2`' as in '`clip`'.

Usage

```
ablineclip(a=NULL,b=NULL,h=NULL,v=NULL,reg=NULL,coef=NULL,untf=FALSE,
  x1=NULL,x2=NULL,y1=NULL,y2=NULL,...)
```

Arguments

`a`	Intercept.
`b`	Slope.
`h`	the x-value(s) for vertical line(s).
`v`	the y-value(s) for horizontal line(s).
`reg`	Fitted lm object.
`coef`	Coefficients, typically intercept and slope.
`untf`	How to plot on log coordinates, see '`abline`'.
`x1,x2,y1,y2`	Clipping limits, see '`clip`'.
`...`	Further arguments passed to '`abline`'.

Details

'`ablineclip`' sets a new clipping region and then calls '`abline`'. If any of the four clipping limits is NULL, the values from '`par("usr")`' are substituted. After the call to '`abline`', the old clipping region is restored. In order to make '`clip`' work, there is a call to '`abline`' that draws a line off the plot.

Value

None. Adds to the current plot.

Author(s)

Remko Duursma

See Also

'`abline`'

Examples

```
x <- rnorm(100)
y <- x + rnorm(100)
lmfit <- lm(y~x)
plot(x,y,xlim=c(-3.5,3.5))
ablineclip(lmfit,x1=-2,x2=2,lty=2)
ablineclip(h=0,x1=-2,x2=2,lty=3,col="red")
ablineclip(v=0,y1=-2.5,y2=1.5,lty=4,col="green")
```

`addtable2plot` *Add a table of values to a plot*

Description

Displays a table of values at a user-specified position on an existing plot

Usage

```
addtable2plot(x,y=NULL,table,lwd=par("lwd"),bty="n",bg=par("bg"),
 cex=1,xjust=0,yjust=1,box.col=par("fg"),text.col=par("fg"),
 display.colnames=TRUE,display.rownames=FALSE,hlines=FALSE,title=NULL)
```

Arguments

`x,y`	Either x and y coordinates to locate the table or an '`xy.coords`' object.
`table`	A data frame, matrix or similar object that will be displayed.
`lwd`	The line width for the box and horizontal dividers.
`bty`	Whether to draw a box around the table ("o") or not ("n").
`bg`	The background color for the table.
`cex`	Character expansion for the table.
`xjust,yjust`	Positioning for the table relative to '`x,y`'.
`box.col`	The color for the box and lines.
`text.col`	The color for the text.
`display.colnames`	Whether to display the column names in the table.
`display.rownames`	Whether to display the row names in the table.
`hlines`	Whether to draw horizontal lines between each row of the table.
`title`	Optional title placed over the table.

Details

'`addtable2plot`' displays the values in '`table`' at a position in user coordinates specified by '`x,y`'. The two justification arguments, '`xjust`' and '`yjust`' are the same as in the '`legend`' function, and '`addtable2plot`' has been programmed to be as similar to '`legend`' as possible. The defaults are those that were most popular in scientific journals at the time of programming.

Value

nil

Author(s)

Original by John Kane, mods by Jim Lemon and Brian Diggs

See Also

'legend'

Examples

```
testdf<-data.frame(Before=c(10,7,5),During=c(8,6,2),After=c(5,3,4))
rownames(testdf)<-c("Red","Green","Blue")
barp(testdf,main="Test addtable2plot",ylab="Value",
 names.arg=colnames(testdf),col=2:4)
# show most of the options
addtable2plot(2,8,testdf,bty="o",display.rownames=TRUE,hlines=TRUE,
 title="The table")
```

`arctext` *Display text on a circular arc.*

Description

'`arctext`' displays a string along a circular arc, rotating each letter. This may not work on all devices, as not all graphic devices can rotate text to arbitrary angles. The output looks best on a Postscript or similar device that can rotate text without distortion. Rotated text often looks very ragged on small bitmaps.

If the user passes a value for '`start`', this will override any value passed to '`middle`'. If the plot area is not square, see '`par(pty="s")`', the arc will be somewhat elliptical.

Usage

```
arctext(x,center=c(0,0),radius=1,start=NA,middle=pi/2,stretch=1,cex=1,...)
```

Arguments

`x`	A character string.
`center`	The center of the circular arc in x/y user units.
`radius`	The radius of the arc in user units.
`start`	The starting position of the string in radians.
`middle`	The middle position of the string in radians.
`stretch`	How much to stretch the string for appearance.
`cex`	The character expansion factor.
`...`	additional arguments passed to '`text`'.

Value

nil

Author(s)

Jim Lemon - Thanks to Suhas Parandekar for the idea.

See Also

'text'

Examples

```
plot(0,xlim=c(1,5),ylim=c(1,5),main="Test of arctext",xlab="",ylab="",
 type="n")
arctext("bendy like spaghetti",center=c(3,3),col="blue")
arctext("bendy like spaghetti",center=c(3,3),radius=1.5,start=pi,cex=2)
arctext("bendy like spaghetti",center=c(3,3),radius=0.5,
 start=pi/2,stretch=1.2)
```

`axis.break` *Place a "break" mark on an axis*

Description

Places a "break" mark on an axis on an existing plot

Usage

```
axis.break(axis=1,breakpos=NULL,bgcol="white",breakcol="black",
 style="slash",brw=0.02)
```

Arguments

`axis`	which axis to break
`breakpos`	where to place the break in user units
`bgcol`	the color of the plot background
`breakcol`	the color of the "break" marker
`style`	Either '`gap`', '`slash`' or '`zigzag`'
`brw`	break width relative to plot width

Value

nil

Note

There is some controversy about the propriety of using discontinuous coordinates for plotting, and thus axis breaks. Discontinuous coordinates allow widely separated groups of values or outliers to appear without devoting too much of the plot to empty space. The major objection seems to be that the reader will be misled by assuming continuous coordinates. The '`gap`' style that clearly separates the two sections of the plot is probably best for avoiding this.

Author(s)

Jim Lemon and Ben Bolker

See Also

'`gap.plot`'

Examples

```
plot(3:10,main="Axis break test")
# put a break at the default axis and position
axis.break()
axis.break(2,2.9,style="zigzag")
twogrp<-c(rnorm(10)+4,rnorm(10)+20)
gap.plot(twogrp,gap=c(8,16),xlab="Index",ylab="Group values",
 main="Two separated groups with gap axis break",
 col=c(rep(2,10),rep(3,10)),ytics=c(3,5,18,20))
legend(12,6,c("Low group","High group"),pch=1,col=2:3)
```

`axis.mult` *Display an axis with values having a multiplier*

Description

An axis is displayed on an existing plot where the tick values are divided by a multiplier and the multiplier is displayed next to the axis.

Usage

```
axis.mult(side=1,at=NULL,labels,mult=1,mult.label,mult.line,
 mult.labelpos=NULL,...)
```

Arguments

`side`	which side to display
`at`	where to place the tick marks - defaults to '`axTicks()`'
`labels`	tick labels - defaults to at/mult
`mult`	the multiplier factor
`mult.label`	the label to show the multiplier - defaults to "x mult"

`mult.line`	the margin line upon which to show the multiplier
`mult.labelpos`	where to place '`mult.label`' - defaults to centered and outside the axis tick labels
`...`	additional arguments passed to '`axis`'.

Details

'`axis.mult`' automates the process of displaying an axis with a multiplier applied to the tick values. By default it will divide the default axis tick labels by '`mult`' and place '`mult.label`' where '`xlab`' or '`ylab`' would normally appear. Thus the plot call should set the relevant label to an empty string in such cases. It is simplest to call '`plot`' with '`axes=FALSE`' and then display the box and any standard axes before calling '`axis.mult`'.

Value

nil

Note

While '`axis.mult`' will try to display an axis on any side, the top and right margins will require adjustment using '`par`' for '`axis.mult`' to display properly.

Author(s)

Jim Lemon

See Also

'`axis`', '`mtext`'

Examples

```
plot(1:10*0.001,1:10*100,axes=FALSE,xlab="",ylab="",main="Axis multipliers")
box()
axis.mult(1,mult=0.001)
axis.mult(2,mult=100)
```

`barp` *A bar plotting routine*

Description

Display a bar plot

Usage

```
barp(height,width=0.4,names.arg=NULL,legend.lab=NULL,legend.pos="e",
col=NULL,border=par("fg"),main=NULL,xlab="",ylab="",xlim=NULL,ylim=NULL,
staxx=FALSE,staxy=FALSE, height.at=NULL,height.lab=NULL,
cex.axis=par("cex.axis"),cylindrical=FALSE,shadow=FALSE,do.first=NULL)
```

Arguments

`height`	A numeric vector, matrix or data frame that will be represented as the heights of bars.
`width`	Half the width of a single bar or group of bars in X axis units.
`names.arg`	The labels for the bars or groups of bars.
`legend.lab`	Labels for an optional legend. If NULL, no legend is displayed.
`legend.pos`	Optional position for the legend as a list with '`x`' and '`y`' components. The default is to call '`emptyspace`' to position the legend. If this is NULL, '`locator`' will be called.
`col`	The fill colors for the bars. The default is no fill.
`border`	The border for the bars.
`main`	The title at the top of the plot.
`xlab,ylab`	The labels for the X and Y axes respectively.
`xlim,ylim`	Optional horizontal and vertical limits for the plot.
`staxx,staxy`	Whether to use '`staxlab`' to stagger the X or Y axis tick labels.
`height.at`	Optional positions of the tick marks on the Y axis.
`height.lab`	Optional tick labels for the Y axis.
`cex.axis`	Character expansion for the axis labels.
`cylindrical`	Whether to give the bars a cylindrical appearance by shading them.
`shadow`	Whether to place a shadow behind the bars.
`do.first`	An optional expression that will be evaluated before anything else is displayed on the plot. Useful for background colors or lines.

Details

'`barp`' displays a bar plot similar to '`barplot`' but with axes and horizontal bar positions more like '`plot`'. Bars or groups of bars are centered on integral X values, and so both the width and spacing of the bars are controlled by a single number. If '`height`' is a vector, single bars representing each value will be displayed centered at '`1:length(height)`'. If '`height`' is a matrix or data frame, a group of bars will be drawn for each column, with the values of the group taken from the rows of that column. The values from '`freq`' or '`brkdn`' in the prettyR package can be used as the '`height`' argument. The value from '`table`' can also be passed as '`height`'.

Bars are empty by default but fill colors can be defined in several ways. If a single color is passed, all bars will be the same color. If '`height`' is a vector, colors will be recycled or some will be ignored if the length of '`col`' is not equal to that of '`height`'. If '`height`' is a matrix or data frame, the user may pass a vector of colors equal to the number of rows in '`height`' or a matrix of colors of the same dimensions as 'height'. Other sequences of color will probably not result in an easy to interpret plot.

'barp' is intended to simplify illustrating categorical data for which both the variable designations and the categories are names, as on many multiple choice questions. 'height.at' and 'height.lab' allow the user to place labels on the vertical axis, usually representing the options. If 'staxx' or 'staxy' are TRUE, the labels on the horizontal or vertical axes respectively will be staggered, allowing the user to use many or lengthy variable or value labels.

'barp' allows two enhancements that may be useful in those areas where fancy plots are appreciated. One is to give the bars a cylindrical look by shading the color. The other is to place an apparent shadow behind each bar. Both of these effects appear as though the light is coming from the upper left, and this is hard coded. You can add error bars by calling 'dispbars', but many advise against this.

If 'legend.lab' is not NULL, a legend will be displayed. If 'legend.pos' is NA, 'locator' is called to place the legend. On Windows, the alert may not appear on the console, and the function will appear to hang unless the user clicks on the console window or the plot.

Value

A list containing two components of the same form as 'height':

x	The centers of the bars displayed.
y	The heights of the bars.

Author(s)

Jim Lemon

See Also

'staxlab', 'barplot', 'cylindrect', 'gradient.rect'

Examples

```
# get some extra room on the left
par(mar=c(5,5,4,2))
# make up some happiness data, as so many seem to do
happyday<-data.frame(Monday=c(2.3,3.4),Tuesday=c(2.8,3.3),Wednesday=c(3.2,3.1),
Thursday=c(3.6,2.8),Friday=c(4.2,2.6),Saturday=c(4.5,2.9),Sunday=c(4.1,2.8))
happylabels<-c("Utterly dashed","Rather mopey","Indifferent","Somewhat elated",
 "Euphoric")
barp(happyday,names.arg=names(happyday),legend.lab=c("Slaves","Unemployed"),
 legend.pos="e",col=c("#ee7700","#3333ff"),main="9AM happiness by weekday",
 xlab="Day of week",ylab="Happiness rating",ylim=c(1,5),staxx=TRUE,staxy=TRUE,
 height.at=1:5,height.lab=happylabels,cex.axis=0.9,cylindrical=TRUE,
 shadow=TRUE)
# now do a plot with colors scaled to the sex ratio (real data!)
# notice how zero and one have been added to get the full proportion range
sexratio<-c(0,1,0.24,0.35,0.09,0.59,0.63,0.34,0.7,0.6)
# the fun ratings are once again a pack of lies
funrating<-c(3.2,3.5,1.5,5.4,4.5,2.7,6.8,4.9)
```

```
funstudy<-c("Astronomy","Chemistry","Economics","Anthropology","Linguistics",
 "Math/Stats","Psychology","Sociology")
funlabels<-c("Torture","Agony","Boredom","Neutral","Entertaining","Exhilarating",
 "Maniacal")
# in the call to color.scale, the leading zero and one are dropped
barp(funrating,names.arg=funstudy,main="Fun ratings for various areas of study",
 col=color.scale(sexratio[-c(1,2)],c(0.2,1),c(0.2,0.4),c(1,0.4)),xlab="Study",
 ylab="Rating",height.at=1:7,height.lab=funlabels,ylim=c(1,7),staxx=TRUE,
 staxy=TRUE,cex.axis=0.9)
# here we want the full scale from zero to one
color.legend(2,6,4,6.4,legend=c("100% guys","100% girls"),
 rect.col=color.scale(seq(0,1,by=0.25),c(0.2,1),c(0.2,0.4),c(1,0.4)))
par(mar=c(5,4,4,2))
# use barp to display a multiple histogram
h1<-table(cut(rnorm(100,4),breaks=seq(0,8,by=2)))
h2<-table(cut(rnorm(100,4),breaks=seq(0,8,by=2)))
h3<-table(cut(rnorm(100,4),breaks=seq(0,8,by=2)))
hmat<-matrix(c(h1,h2,h3),nrow=3,byrow=TRUE)
barp(hmat,names.arg=names(h1),width=0.45,col=2:4,
 main="Multiple histogram using barp",xlab="Bins",ylab="Frequency")
legend(3.8,50,c("h1","h2","h3"),fill=2:4)
```

bin.wind.records *Classify wind direction and speed records.*

Description

Classifies wind direction and speed records into a matrix of percentages of observations in speed and direction bins.

Usage

```
bin.wind.records(winddir,windspeed,ndir=8,radians=FALSE,
 speed.breaks=c(0,10,20,30))
```

Arguments

winddir	A vector of wind directions.
windspeed	A vector of wind speeds corresponding to the above directions.
ndir	Number of direction bins in a compass circle.
radians	Whether wind directions are in radians.
speed.breaks	Minimum wind speed for each speed bin.

Details

'`bin.wind.records`' bins a number of wind direction and speed records into a matrix of percentages of observations that can be used to display a cumulative wind rose with '`oz.windrose`' The defaults are those used by the Australian Bureau of Meteorology.

Value

A matrix of percentages in which the rows represent wind speed categories and the columns represent wind direction categories.

Author(s)

Jim Lemon

See Also

'`oz.windrose`'

Examples

```
winddir<-sample(0:360,100,TRUE)
windspeed<-sample(0:40,100,TRUE)
bin.wind.records(winddir,windspeed)
```

`boxed.labels` *Place labels in boxes*

Description

Places labels in boxes on an existing plot

Usage

```
boxed.labels(x,y=NA,labels,
 bg=ifelse(match(par("bg"),"transparent",0),"white",par("bg")),
 border=TRUE,xpad=1.2,ypad=1.2,srt=0,cex=1,adj=0.5,...)
```

Arguments

`x,y`	x and y position of the centers of the labels. '`x`' can be an '`xy.coords`' list.
`bg`	The fill color of the rectangles on which the labels are displayed (see Details).
`labels`	Text strings
`border`	Whether to draw borders around the rectangles.
`xpad,ypad`	The proportion of the rectangles to the extent of the text within.
`srt`	Rotation of the labels. If 90 or 270 degrees, the box will be rotated 90 degrees.
`cex`	Character expansion. See '`text`'.
`adj`	left/right adjustment. If this is set outside the function, the box will not be aligned properly.
`...`	additional arguments passed to '`text`'.

Details

The label(s) are displayed on a rectangular background. This may be useful for visibility and is the reason that "transparent" background is not available. Only right angle rotations are allowed in ‘`boxed.labels`’. *Important change*: ‘`xpad`’ and ‘`ypad`’ are now the full proportion of the box to text, not half. The user can now call ‘`cylindrect`’ or ‘`gradient.rect`’ for the background rectangle.

Value

nil

Note

This function is best for regularly spaced labels where overlapping is not a problem. See ‘`thigmophobe.labels`’ for placing labels where overlap is likely.

Author(s)

Jim Lemon

See Also

‘`spread.labels`’, ‘`thigmophobe.labels`’

Examples

```
x<-rnorm(10)
y<-rnorm(10)
plot(x,y,type="p")
nums<-c("one","two","three","four","five","six","seven","eight","nine","ten")
boxed.labels(x,y-0.1,nums)
# now label a barplot
xpos<-barplot(c(1,3,2,4))
boxed.labels(xpos,0.5,nums[1:4])
# perform a PCA on the "swiss" dataset and plot the first two components
data(swiss)
swiss.pca<-prcomp(swiss)
plot(swiss.pca$rotation[,1:2],xlim=c(-1,0.2),main="PCA of swiss dataset",
 type="n")
boxed.labels(swiss.pca$rotation[1:6],swiss.pca$rotation[7:12],ypad=1.5,
 colnames(swiss),bg=c("red","purple","blue","blue","darkgreen","red"),
 col="white")
```

brkdn.plot	*A point/line plotting routine*

Description

Display a point/line plot of breakdowns of one or more variables.

Usage

```
brkdn.plot(vars,groups=NA,obs=NA,data,mct="mean",md="std.error",stagger=NA,
dispbar=TRUE,main="Breakdown plot",xlab=NA,ylab=NA,xaxlab=NA,
ylim=NA,type="b",pch=1,lty=1,col=par("fg"),staxx=FALSE,...)
```

Arguments

`vars`	The names or indices of one or more columns in a data frame. The columns must contain numeric data.
`groups`	The name or index of a column in a data frame that classifies the values in '`vars`' into different, usually fixed effect, levels.
`obs`	The name or index of a column in a data frame that classifies the values in '`vars`' into different, usually random effect, levels.
`data`	The data frame.
`mct`	The measure of central tendency to calculate for each group.
`md`	The measure of dispersion to calculate, NA for none.
`stagger`	The amount to offset the successive values at each horizontal position as a proportion of the width of the plot. The calculated default is usually adequate. Pass zero for none.
`dispbar`	Whether to display the measures of dispersion as bars.
`main`	The title at the top of the plot.
`xlab,ylab`	The labels for the X and Y axes respectively. There are defaults, but they are basic.
`xaxlab`	Optional labels for the horizontal axis ticks.
`ylim`	Optional vertical limits for the plot.
`type`	Whether to plot symbols, lines or both (as in '`plot`').
`pch`	Symbol(s) to plot.
`lty`	Line type(s) to plot.
`col`	Color(s) for the symbols and lines.
`staxx`	Whether to call '`staxlab`' to display the X axis labels.
`...`	additional arguments passed to '`plot`'.

Details

'brkdn.plot' displays a plot useful for visualizing the breakdown of a response measure by two factors, or more than one response measure by either a factor representing something like levels of treatment ('groups') or something like repeated observations ('obs'). For example, if observations are made at different times on data objects that receive different treatments, the 'groups' factor will display the measures of central tendency as points/lines with the same color, symbol and line type, while the 'obs' factor will be represented as horizontal positions on the plot. If 'obs' is numeric, its unique values will be used as the positions, if not, 1 to the number of unique values. This is a common way of representing changes over time intervals for experimental groups.

Value

A list of two matrices of dimension 'length(levels(groups))' by 'length(levels(obs))'. The first contains the measures of central tendency calculated and its name is the name of the function passed as 'mct'. The second contains the measures of dispersion and its name is the name of the function passed as 'md'.

If both 'groups' and 'obs' are not NA, the rows of each matrix will be the 'groups' and the columns the 'obs'. If 'obs' is NA, the rows will be the 'groups' and the columns the 'vars'. If 'groups' is NA, the rows will be the 'vars' and the columns the 'obs'. That is, if 'vars' has more than one element, if 'obs' is NA, the elements of 'vars' will be considered to represent observations, while if 'groups' is NA, they will be considered to represent groups. At least one of 'groups' and 'obs' must be not NA or there is no point in using 'brkdn.plot'.

Author(s)

Jim Lemon

See Also

'dispbars'

Examples

```
test.df<-data.frame(a=rnorm(80)+4,b=rnorm(80)+4,c=rep(LETTERS[1:4],each=20),
 d=rep(rep(letters[1:4],each=4),5))
# first use the default values
brkdn.plot("a","c","d",test.df,pch=1:4,col=1:4)
# now jazz it up a bit using medians and median absolute deviations
# and some enhancements
bp<-brkdn.plot("a","c","d",test.df,main="Test of the breakdown plot",
 mct="median",md="mad",xlab="Temperature range", ylab="Cognition",
 xaxlab=c("10-15","16-20","21-25","25-30"),pch=1:4,lty=1:4,col=1:4)
es<-emptyspace(bp)
legend(es,legend=c("Sydney","Gosford","Karuah","Brisbane"),pch=1:4,
 col=1:4,lty=1:4,xjust=0.5,yjust=0.5)
```

bumpchart *Display a "bumps" chart.*

Description

Display a chart with two of more columns of points in order of ascending values with lines connecting the points in a row.

Usage

```
bumpchart(y,top.labels=colnames(y),labels=rownames(y),rank=TRUE,
 mar=c(2,8,5,8),pch=19,col=par("fg"),lty=1,lwd=1,...)
```

Arguments

`y`	A numeric matrix or data frame which may contain NAs.
`top.labels`	The strings that will appear at the top of each column of points on the plot.
`labels`	The strings that will appear next to the outer columns of points.
`rank`	Whether to rank the values in '`y`' before plotting.
`mar`	The margins to use for the bumps chart.
`pch`	The symbols to use when plotting the points.
`col`	The colors to use.
`lty`	The line types to use.
`lwd`	The line widths to use.
`...`	Additional arguments passed to '`matplot`'.

Details

'`bumpchart`' calls '`matplot`' to plot the values in the transposed '`y`' matrix or data frame, joining the points with lines. At the left and right edges of the plot, the labels identifying each row of points are displayed. This type of plot is often used to show the changing positions of entities over time, like the ranking in surveys in different years.

Because of the way '`matplot`' plots the values, the order of everything is reversed. If the user wants to rank in descending order, such as test scores, rank the values in the correct order before sending them to '`bumpchart`'.

Value

nil

Author(s)

Jim Lemon

See Also

'matplot'

Examples

```
# percentage of those over 25 years having completed high school
# in 10 cities in the USA in 1990 and 2000
educattn<-matrix(c(90.4,90.3,75.7,78.9,66,71.8,70.5,70.4,68.4,67.9,
 67.2,76.1,68.1,74.7,68.5,72.4,64.3,71.2,73.1,77.8),ncol=2,byrow=TRUE)
rownames(educattn)<-c("Anchorage AK","Boston MA","Chicago IL",
 "Houston TX","Los Angeles CA","Louisville KY","New Orleans LA",
 "New York NY","Philadelphia PA","Washington DC")
colnames(educattn)<-c(1990,2000)
bumpchart(educattn,main="Rank for high school completion by over 25s")
# now show the raw percentages and add central ticks
bumpchart(educattn,rank=FALSE,
 main="Percentage high school completion by over 25s",col=rainbow(10))
# margins have been reset, so use
par(xpd=TRUE)
boxed.labels(1.5,seq(65,90,by=5),seq(65,90,by=5))
par(xpd=FALSE)
```

`centipede.plot` *Display a centipede plot*

Description

Displays a centipede plot on the current graphics device.

Usage

```
centipede.plot(segs,mct="mean",lower.limit="std.error",
 upper.limit=lower.limit,left.labels=NULL,right.labels=NULL,sort.segs=TRUE,
 main="",xlab=NA,vgrid=NA,mar=NA,col=par("fg"),bg="green",...)
```

Arguments

`segs`	a matrix of midpoints and limits calculated by 'get.segs' OR a 'dstat' object returned by 'brkdn'.
`mct`	The function to use in calculating the midpoint of each segment.
`lower.limit`	The functions to use in calculating the lower limits for each subset of the data.
`upper.limit`	The functions to use in calculating the upper limits.
`left.labels`	The variable or subset labels to place at the left margin of the plot. Default values are provided.
`right.labels`	The variable or subset labels to place at the right margin of the plot.
`sort.segs`	Whether to sort the segments in ascending order.

`main`	Optional title for the plot.
`xlab`	Optional x axis label for the plot. The default NA displays a text label showing the midpoint and limit functions.
`vgrid`	Optional vertical line(s) to display on the plot.
`mar`	Margin widths for the plot. Defaults to c(4,5,1,4) or c(4,5,3,4) if there is a title.
`col`	The color(s) of the limit lines and borders of the midpoint markers.
`bg`	The color(s) to fill the midpoint markers.
`...`	additional arguments passed to '`plot`'.

Details

'`centipede.plot`' displays one or more midpoints and limits as filled circles with horizontal error bars. It places labels on the left and right sides of the plot. If these labels are long, it may be necessary to pass explicit values to the '`mar`' argument to leave enough room.

Similarly, centipede plots typically have a large number of subsets, and it may be necessary to start the graphics device with an aspect ratio that will prevent crowding of the labels when over 30 segments are displayed.

The matrix '`segs`' may be entered manually or read from a file. The first row specifies midpoints, the second and third rows the lower and upper limits respectively and the fourth row the number of valid observations. If a '`dstat`' object is passed as '`segs`', the function will calculate the lower and upper values according to the relevant arguments. This type of plot is also known as a caterpillar plot or a league table.

Value

nil.

Author(s)

Jim Lemon

See Also

'`get.segs`'

Examples

```
testcp<-list("",40)
for(i in 1:40) testcp[[i]]<-rnorm(sample(1:8,1)*50)
segs<-get.segs(testcp)
centipede.plot(segs,main="Test centipede plot",vgrid=0)
```

clean.args — *Remove inappropriate arguments from an argument list*

Description

Takes a list of arguments and eliminates those that are not appropriate for passing to a particular function (and hence would produce an error if passed).

Usage

```
clean.args(argstr,fn,exclude.repeats=FALSE,exclude.other=NULL,dots.ok=TRUE)
remove.args(argstr,fn)
```

Arguments

`argstr`	a named list of arguments, e.g. from ‘`...`’
`fn`	a function
`exclude.repeats`	(logical) remove repeated arguments?
`exclude.other`	a character vector of names of additional arguments to remove
`dots.ok`	should "..." be allowed in the argument list?

Value

‘`clean.args`’ returns a list which is a copy of ‘`argstr`’ with arguments inappropriate for ‘`fn`’ removed; ‘`remove.args`’ removes the arguments for ‘`fn`’ from the list.

Author(s)

Ben Bolker

Examples

```
tststr <- list(n=2,mean=0,sd=1,foo=4,bar=6)
clean.args(tststr,rnorm)
try(do.call("rnorm",tststr))
do.call("rnorm",clean.args(tststr,rnorm))
remove.args(tststr,rnorm)
## add example of combining arg. lists?
```

`clock24.plot` *Plot values on a 24 hour "clockface".*

Description

'`clock24.plot`' displays a plot of radial lines, symbols or a polygon centered at the midpoint of the plot frame on a 24 hour 'clockface'. In contrast to the default behavior of '`radial.plot`', the positions are interpreted as beginning at vertical (000) and moving clockwise.

Usage

```
clock24.plot(lengths,clock.pos,labels=NULL,label.pos=NULL,rp.type="r",...)
```

Arguments

`lengths`	numeric data vector. Magnitudes will be represented as line lengths, or symbol or polygon vertex positions.
`clock.pos`	numeric vector of positions on the 'clockface'. These must be in decimal hours and will be rescaled to radians.
`labels`	Labels to place at the circumference.
`label.pos`	Radial positions of the labels.
`rp.type`	Whether to plot radial lines, symbols or a polygon.
`...`	additional arguments are passed to '`radial.plot`' and then to '`plot`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`polar.plot`','`radial.plot`'

Examples

```
testlen<-rnorm(24)*2+5
testpos<-0:23+rnorm(24)/4
clock24.plot(testlen,testpos,main="Test Clock24 (lines)",show.grid=FALSE,
 line.col="green",lwd=3)
if(dev.interactive()) par(ask=TRUE)
# now do a 'daylight' plot
clock24.plot(testlen[7:19],testpos[7:19],
 main="Test Clock24 daytime (symbols)",
 point.col="blue",rp.type="s",lwd=3)
```

```
# reset the margins
par(mar=c(5,4,4,2))
```

`clplot` *Plot lines with colors determined by values.*

Description

'`clplot`' displays a plot of lines for which the colors are dependent upon the x and y values. '`clplot`' is similar to '`color.scale.lines`' except that while the latter calculates a color for each unique value, '`clplot`' assigns colors to groups of values within the cutpoints defined by '`levels`'.

Usage

```
clplot(x,y,ylab=deparse(substitute(y)),xlab=deparse(substitute(x)),
 levels=seq(min(y)+(max(y)-min(y))/5,max(y)-(max(y)-min(y))/5,length.out=4),
 cols=c("black","blue","green","orange","red"),lty=1,showcuts=FALSE,...)
```

Arguments

`x,y`	numeric data vectors.
`ylab,xlab`	Labels for the X and Y axes.
`levels`	Cut points to assign colors to the values of '`x`' and '`y`'.
`cols`	The colors to be assigned.
`lty`	The line type.
`showcuts`	Whether to show the positions of the cut points.
`...`	additional arguments passed to '`plot`' or '`lines`'.

Value

nil

Author(s)

Carl Witthoft

See Also

'`plot`'

Examples

```
x<-seq(1,100)
y<-sin(x/5)+x/20
clplot(x,y,main="Test of clplot")
```

`cluster.overplot` *Shift overlying points into clusters.*

Description

'`cluster.overplot`' checks for overlying points in the x and y coordinates passed. Those points that are overlying are moved to form a small cluster of up to nine points. For large numbers of overlying points, see '`count.overplot`' or '`sizeplot`'. If you are unsure of the number of overplots in your data, run '`count.overplot`' first to see if there are any potential clusters larger than nine.

Usage

```
cluster.overplot(x,y,away=NULL,tol=NULL,...)
```

Arguments

`x,y`	Numeric data vectors or the first two columns of a matrix or data frame. Typically the x/y coordinates of points to be plotted.
`away`	How far to move overlying points in user units. Defaults to the width of a lower case "o" in the x direction and 5/8 of the height of a lower case "o" in the y direction.
`tol`	The largest distance between points that will be considered to be overlying. Defaults to 1/2 of the width of a lower case "o" in the x direction and 1/2 of the height of a lower case "o" in the y direction.
`...`	additional arguments returned as they are passed.

Value

A list with two components. For unique x-y pairs the elements will be the same as in the original. For overlying points up to eight additional points will be generated that will create a cluster of points instead of one.

Author(s)

Jim Lemon

See Also

'`count.overplot`','`sizeplot`'

Examples

```
xy.mat<-cbind(sample(1:10,200,TRUE),sample(1:10,200,TRUE))
clusteredpoints<-
 cluster.overplot(xy.mat,col=rep(c("red","green"),each=100))
plot(clusteredpoints,col=clusteredpoints$col,
 main="Cluster overplot test")
```

color.gradient *Calculate an arbitrary sequence of colors.*

Description

'`color.gradient`' is now just a call to '`color.scale`' with a vector of equally spaced integers (1:nslices). The function is kept for backward compatibility.

Usage

```
color.gradient(reds,greens,blues,nslices=50)
```

Arguments

`reds,greens,blues`	vectors of the values of the color components as 0 to 1.
`nslices`	The number of color "slices".

Value

A vector of hexadecimal color values as used by '`col`'.

Note

The function is mainly useful for defining a set of colors to represent a known number of gradations. Such a set can be used to assign a grade to a small number of values (e.g. points on a scatterplot - but see '`color.scale`' for large numbers) and display a color bar using '`gradient.rect`' as a legend.

Author(s)

Jim Lemon

See Also

'`rescale`','`approx`','`color.scale`'

Examples

```
# try it with red and blue endpoints and green midpoints.
color.gradient(c(0,1),c(1,0.6,0.4,0.3,0),c(0.1,0.6))
```

`color.id` *Identify closest match to a color*

Description

Given a color given as a hex string, find the closest match in the table of known (named) colors

Usage

```
color.id(col)
```

Arguments

`col`	a color specified as a hex string

Details

finds the color with the minimum squared distance in RGB space

Value

the name of the closest match

Author(s)

Ben Bolker

See Also

‘`col2rgb`’,‘`colors`’

Examples

```
color.id("#cc00cc")
```

`color.legend` *Legend matching categories or values to colors*

Description

Display a color legend on a plot

Usage

```
color.legend(xl,yb,xr,yt,legend,rect.col,cex=1,align="lt",gradient="x",...)
```

Arguments

`xl,yb,xr,yt`	The lower left and upper right coordinates of the rectange of colors in user coordinates.
`legend`	The labels that will appear next to some or all of the colors.
`rect.col`	The colors that will fill the rectangle.
`cex`	Character expansion factor for the labels.
`align`	How to align the labels relative to the color rectangle.
`gradient`	Whether to have a horizontal (x) or vertical (y) color gradient.
`...`	Additional arguments passed to '`text`'.

Details

'`color.legend`' displays a rectangle defined by the first four arguments filled with smaller rectangles of color defined by the '`rect.col`' argument. Labels, defined by the '`legend`' argument, are placed next to the color rectangle. The position of the labels is determined by whether the color rectangle is horizontal or vertical and the '`align`' argument. The default value of '`lt`' places the labels at the left of a vertical rectangle or the top of a horizontal one. '`rb`' puts them on the other side. To have the labels in the same color as the rectangles, include a '`col`' argument that will be passed to '`text`' as in the example.

There can be fewer labels than colors. The labels will be evenly spaced along the rectangle in this case. It is possible to use empty labels to get uneven spacing. The user can pass more labels than colors, but the labels will almost certainly be crowded and it is not obvious that this would be of any use. To have complete control over the labels, see '`gradient.rect`' and '`text`' or '`mtext`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`color.gradient`', '`gradient.rect`'

Examples

```
# get some extra room
par(mar=c(7,4,4,6))
testcol<-color.gradient(c(0,1),0,c(1,0),nslices=5)
col.labels<-c("Cold","Warm","Hot")
color2D.matplot(matrix(rnorm(100),nrow=10),c(1,0),0,c(0,1),
 main="Test color legends")
color.legend(11,6,11.8,9,col.labels,testcol,gradient="y")
color.legend(10.2,2,11,5,col.labels,testcol,align="rb",gradient="y")
color.legend(0.5,-2,3.5,-1.2,col.labels,testcol)
color.legend(7,-1.8,10,-1,col.labels,testcol,align="rb",col=testcol[c(1,3,5)])
par(mar=c(5,4,4,2))
```

`color.scale` *Turn values into colors.*

Description

Transform numeric values into colors.

Usage

```
color.scale(x,redrange=c(0,1),greenrange=c(0,1),bluerange=c(0,1),
 extremes=NA,na.color=NA)
```

Arguments

`x`	a numeric vector, matrix or data frame
`redrange,greenrange,bluerange`	color ranges into which to scale '`x`'
`extremes`	The colors for the extreme values of '`x`'.
`na.color`	The color to use for NA values of '`x`'.

Details

'`color.scale`' calculates a sequence of colors by a linear transformation of the numeric values supplied into the ranges for red, green and blue. If only one number is supplied for a color range, that color remains constant for all values of '`x`'. If more than two values are supplied, the '`x`' values will be split into equal ranges (one less than the number of colors) and the transformation carried out on each range. Values for a color range must be between 0 and 1.

If '`extremes`' is not NA, the ranges will be calculated from its values using '`col2rgb`', even if ranges are also supplied. '`extremes`' allows the user to just pass the extreme color values in any format that '`col2rgb`' will accept.

The user may not want the color scheme to be continuous across some critical point, often zero. In this case, color scale can be called separately for the values below and above zero. See the second example for '`color2D.matplot`'.

Value

A vector or matrix of hexadecimal color values.

Note

The function is useful for highlighting a numeric dimension or adding an extra "dimension" to a plot.

Author(s)

Jim Lemon

See Also

'rescale', 'col2rgb'

Examples

```
# go from green through yellow to red with no blue
x<-rnorm(20)
y<-rnorm(20)
# use y for the color scale
plot(x,y,col=color.scale(y,c(0,1,1),c(1,1,0),0),main="Color scale plot",
 pch=16,cex=2)
```

`color.scale.lines` *Line segments with scaled colors.*

Description

Display line segments with colors scaled to numeric values.

Usage

```
color.scale.lines(x,y,reds,greens,blues,col=NA,colvar=NA,...)
```

Arguments

`x,y`	Numeric vectors or a list with at least two components, the first two of which must be named x and y.
`reds,greens,blues`	Color ranges into which to scale the numeric values.
`col`	One or more colors to use for the resultant lines. Will be recycled if necessary.
`colvar`	A numeric vector from which to scale the colors.
`...`	Additional arguments passed to '`segments`'.

Details

'`color.scale.lines`' displays line segments that can be individually colored according to a variety of methods. In order of precedence, if '`col`' is not NA, the color values passed will be used. If '`colvar`' is not NA, the function will call '`color.scale`' with the three color range arguments to determine the line colors. If '`colvar`' is the same length as '`length(x)-1`', exactly enough colors for the number of lines displayed will be calculated. If shorter, some colors will be recycled and if longer, some colors will not be used. Finally, the values in '`y`' will be color-scaled if both of the above arguments are NA. Thus the user can pass predetermined colors, use colors scaled from an arbitrary vector of numerical values or use the '`y`' values. See '`color.scale`' for an explanation of specifying color ranges.

Value

nil

Note

The function is useful for highlighting a numeric dimension or adding an extra "dimension" to a plot.

Author(s)

Jim Lemon

See Also

'`color.scale`'

Examples

```
# color a random walk "hot" (red) to "cold" (blue) on its distance
# from the starting point
x<-c(0,cumsum(rnorm(99)))
y<-c(0,cumsum(rnorm(99)))
xydist<-sqrt(x*x+y*y)
plot(x,y,main="Random walk plot",xlab="X",ylab="Y",type="n")
color.scale.lines(x,y,c(1,1,0),0,c(0,1,1),colvar=xydist,lwd=2)
boxed.labels(x,y,labels=1:100,border=FALSE,cex=0.5)
```

`color2D.matplot` *Display a numeric matrix as color matrix*

Description

Display the values of a numeric 2D matrix or data frame as colored rectangles or hexagons.

Usage

```
color2D.matplot(x,redrange=c(0,1),greenrange=c(0,1),bluerange=c(0,1),
 extremes=NA,cellcolors=NA,show.legend=FALSE,nslices=10,xlab="Column",
 ylab="Row",do.hex=FALSE,axes=TRUE,show.values=FALSE,vcol="white",vcex=1,
 border="black",na.color=NA,...)
```

Arguments

`x`	data values
`redrange, greenrange, bluerange`	the ranges of red, green and blue that will be scaled to represent the range of numeric values
`extremes`	The colors for the extreme values of '`x`'. Takes precedence over the color ranges.

`cellcolors`	A precalculated vector of cell colors. This must have exactly the same number of colors as there are values in the matrix or it will be uninformative. Takes precedence over both '`extremes`' and color ranges.
`show.legend`	whether to display a color legend with the extreme numeric values in the lower left corner of the plot. If the default is not suitable, call '`color.legend`' separately.
`nslices`	The number of color "slices" in the legend.
`xlab,ylab`	axis labels for the plot.
`do.hex`	plot packed hexagons instead of rectangles.
`axes`	Whether to suppress the default axis labelling.
`show.values`	Whether to display the numeric values of '`x`'. This also controls the number of decimal places displayed.
`vcol`	The color for the value display.
`vcex`	The character expansion for the value display.
`border`	The color(s) for the borders of the cells. Pass NA if no border is wanted.
`na.color`	The color to use for NA values of '`x`'.
`...`	arguments passed to '`plot`'.

Details

Displays a plot with the same number of rectangular or hexagonal cells as there are numeric values in the matrix or data frame. Each rectangle is colored to represent its corresponding value. The rectangles are arranged in the conventional display of a 2D matrix with rows beginning at the top and columns at the left. The color scale defaults to black for the minimum value and white for the maximum.

The user will have to adjust the plot device dimensions to get regular squares or hexagons, especially when the matrix is not square. As the margins are not equivalent for all display devices, this is currently a matter of trial and error. Drawing hexagons is quite slow.

'`show.values`' is also used to control the number of decimal places displayed if the values are shown. '`TRUE`' will give one decimal place, '`2`' two, and so on.

Value

nil

Note

The function '`image`' performs almost the same when passed a matrix of values without grid positions, except that it assigns values to a specified list of colors rather than calculating a color for each distinct value.

Author(s)

Jim Lemon (thanks to Ashoka Polpitiya for '`axes`')

See Also

'color.scale', 'image'

Examples

```
x<-matrix(rnorm(1024)+sin(seq(0,2*pi,length=1024)),nrow=32)
color2D.matplot(x,c(1,0),c(0,0),c(0,1),show.legend=TRUE,
 xlab="Columns",ylab="Rows",main="2D matrix plot")
# generate colors that show negative values in red and positive in green
cellcol<-matrix(rep("#000000",1024),nrow=32)
cellcol[x<0]<-color.scale(x[x<0],c(1,0.8),c(0,0.8),0)
cellcol[x>0]<-color.scale(x[x>0],0,c(0.8,1),c(0.8,0))
# now do hexagons without borders
color2D.matplot(x,cellcolors=cellcol,xlab="Columns",ylab="Rows",
 do.hex=TRUE,main="2D matrix plot (hexagons)",border=NA)
# for this one, we have to do the color legend separately
# because of the two part color scaling
legval<-seq(min(x),max(x),length.out=6)
legcol<-rep("#000000",6)
legcol[legval<0]<-color.scale(legval[legval<0],c(1,0.8),c(0,0.8),0)
legcol[legval>0]<-color.scale(legval[legval>0],0,c(0.8,1),c(0.8,0))
color.legend(0,-5,6,-4,round(c(min(x),0,max(x)),1),rect.col=legcol)
# do a color only association plot
xt<-table(sample(1:10,100,TRUE),sample(1:10,100,TRUE))
observed<-xt[,rev(1:dim(xt)[2])]
expected<-outer(rowSums(observed),colSums(observed),"*")/sum(xt)
deviates<-(observed-expected)/sqrt(expected)
cellcol<-matrix(rep("#000000",100),nrow=10)
cellcol[deviates<0]<-
 color.scale(deviates[deviates<0],c(1,0.8),c(0,0.5),0)
cellcol[deviates>0]<-
 color.scale(deviates[deviates>0],0,c(0.7,0.8),c(0.5,0))
color2D.matplot(x=round(deviates,2),cellcolors=cellcol,
 show.values=TRUE,main="Association plot")
```

---

convert.meq   *Convert milligrams/liter to milliequivalents/liter.*

Description

Convert milligrams/liter of ions used in the Piper diagram to milliequivalents/liter.

Usage

```
convert.meq(mgpl,parameter)
```

Arguments

`mgpl`	A data frame containing the concentrations of ions used in the Piper diagram in milligrams/liter.
`parameter`	The names of the ions if necessary.

Details

‘`convert.meq`’ does the conversion with a lookup table of molecular weights and valences.

Value

The concentration of the ions in milliequivalents.

Author(s)

Mike Cheetham

See Also

‘`piper.diagram`’

`corner.label` *Find corner locations and optionally display a label*

Description

Finds the coordinates in user parameters of a specified corner of the figure region and optionally displays a label there

Usage

```
corner.label(label=NULL,x=-1,y=1,xoff=NA,yoff=NA,figcorner=FALSE,...)
```

Arguments

`label`	Text to display. The default is to display nothing.
`x`	an integer value: -1 for the left side of the plot, 1 for the right side
`y`	an integer value: -1 for the bottom side of the plot, 1 for the top side
`xoff,yoff`	Horizontal and vertical text offsets. Defaults to one half of the width and height of "m" respectively.
`figcorner`	Whether to find/display at the corner of the plot or figure.
`...`	further arguments to the ‘`text`’ command for the label

Details

'`corner.label`' finds the specified corner of the plot or figure and if '`label`' is not NULL, displays it there. The text justification is specified so that the label will be justified away from the corner. To get the label squeezed right into a corner, set '`xoff`' and '`yoff`' to zero.

Value

A list of the x and y positions of the corner adjusted for the offsets.

Author(s)

Ben Bolker

Examples

```
plot(1:10,1:10)
corner.label("A")
corner.label(x=1,y=1)
corner.label("B",y=-1,x=1,figcorner=TRUE,col="red")
```

`count.overplot` *Show overlying points as counts*

Description

'`count.overplot`' checks for overlying points defined as points separated by a maximum of '`tol`', a two element numeric vector of the x and y tolerance. Defaults to 1/2 of the width of a lower case "o" in the x direction and 1/2 of the height of a lower case "o" in the y direction.

Usage

```
count.overplot(x,y,tol=NULL,col=par("fg"),pch="1",...)
```

Arguments

`x,y`	Two numeric data vectors or the first two columns of a matrix or data frame. Typically the x/y coordinates of points to be plotted.
`tol`	The largest distance between points that will be considered to be overlying.
`col`	Color(s) for the points (not the numbers).
`pch`	Symbol(s) to display.
`...`	additional arguments passed to '`plot`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`cluster.overplot`','`sizeplot`'

Examples

```
xy.mat<-cbind(sample(1:10,200,TRUE),sample(1:10,200,TRUE))
count.overplot(xy.mat,main="count.overplot",
 xlab="X values",ylab="Y values")
```

`cylindrect` *Display an apparent cylinder*

Description

Display rectangles shaded to appear like cylinders.

Usage

```
cylindrect(xleft,ybottom,xright,ytop,col,border=NA,gradient="x",nslices=50)
```

Arguments

`xleft`	The position of the left side of the rectangle(s).
`ybottom`	The position of the bottom of the rectangle(s).
`xright`	The position of the right side of the rectangle(s).
`ytop`	The position of the top side of the rectangle(s).
`col`	The base color(s) of the rectangles.
`border`	Whether to draw a border and what color.
`gradient`	Whether to vary the shading horizontally ("x" - the default) or vertically (anything but "x").
`nslices`	The number of "slices" of color for shading.

Details

'`cylindrect`' displays a rectangle filled with "slices" of color that simulate the appearance of a cylinder. The slices are calculated so that the base color appears at the right or bottom edge of the rectangle, become progressively lighter to a "highlight" at two thirds of the width or height and then darken toward the base color again.

The appearance is of a cylinder lit from above and to the left of the viewer. The position of the apparent light source is hard coded into the function.

Value

The base color(s) of the rectangle(s).

Author(s)

Jim Lemon

See Also

‘gradient.rect’

Examples

```
plot(0,xlim=c(0,5),ylim=c(0,5),main="Examples of pseudocylindrical rectangles",
 xlab="",ylab="",axes=FALSE,type="n")
cylindrect(0,0,1,5,"red")
cylindrect(rep(1,3),c(0,2,4),rep(4,3),c(1,3,5),"green",gradient="y")
cylindrect(4,0,5,5,"#8844aa")
```

---

diamondplot — *Plot multiple variables as polygons on a radial grid.*

Description

‘diamondplot’ displays a plot of polygons on a radial grid representing the relationships between one or more attributes of data objects. For a slightly different style of plot, see the "spiderweb plot" example in ‘radial.plot’.

Usage

```
diamondplot(x, bg=gray(0.6), col=rainbow,name="", ...)
```

Arguments

x	A data frame containing numeric values that represent attributes (possibly repeated observations) of data objects. See the example.
bg	The background color for the plot.
col	The colors for the polygons.
name	The title for the plot (i.e. ‘main’).
...	additional arguments passed to ‘plot’.

Value

nil

Author(s)

Elisa Biancotto

See Also

‘plot’, ‘radial.plot’

Examples

```
data(mtcars)
mysubset<-mtcars[substr(dimnames(mtcars)[[1]],1,1)=="M",c("mpg","hp","wt","disp")]
diamondplot(mysubset)
```

---

dispersion *Display a measure of dispersion*

Description

Display line/cap bars at specified points on a plot representing measures of dispersion.

Usage

```
dispersion(x,y,ulim,llim=ulim,intervals=TRUE,arrow.cap=0.01,arrow.gap=NA,
 type="a",fill=NA,...)
dispbars(x,y,ulim,llim=ulim,intervals=TRUE,arrow.cap=0.01,arrow.gap=NA,...)
```

Arguments

`x,y`	x and y position of the centers of the bars
`ulim,llim`	The extent of the dispersion measures.
`arrow.cap`	The width of the cap at the outer end of each bar as a proportion of the width of the plot.
`arrow.gap`	The gap to leave at the inner end of each bar. Defaults to two thirds of the height of a capital "O".
`intervals`	Whether the limits are intervals (TRUE) or absolute values (FALSE).
`type`	What type of display to use.
`fill`	Color to fill between the lines if ‘`type`’ is not NULL and ‘`fill`’ is not NA.
`...`	additional arguments passed to ‘`arrows`’ or ‘`lines`’ depending upon ‘`type`’.

Details

'dispersion' displays a measure of dispersion on an existing plot. Currently it will display either vertical lines with caps (error bars) or lines that form a "confidence band" around a line of central tendency. If 'fill' is not NA and 'type' is '"l"', a polygon will be drawn between the confidence lines. Remember that any points or lines within the confidence band will be obscured, so these will have to be redrawn.

The 'intervals' argument allows the user to pass the limits as either intervals (the default) or absolute values.

'dispbars' is now a call to 'dispersion' with 'type' set to 'a'.

If 'arrow.gap' is greater than or equal to the upper or lower limit for a bar, 'segments' is used to draw the upper and lower caps with no bars to avoid zero length arrows.

Value

nil

Author(s)

Jim Lemon

See Also

'arrows', 'segments','lines'

Examples

```
disptest<-matrix(rnorm(200),nrow=20)
disptest.means<-rowMeans(disptest)
row.order<-order(disptest.means)
se.disptest<-unlist(apply(disptest,1,std.error))
plot(disptest.means[row.order],main="Dispersion as error bars",
 ylim=c(min(disptest.means-se.disptest),max(disptest.means+se.disptest)),
 xlab="Occasion",ylab="Value")
dispersion(1:20,disptest.means[row.order],se.disptest[row.order])
plot(disptest.means[row.order],main="Dispersion as confidence band",
 ylim=c(min(disptest.means-se.disptest),max(disptest.means+se.disptest)),
 xlab="Occasion",ylab="Value")
dispersion(1:20,disptest.means[row.order],se.disptest[row.order],type="l",
 fill="#eeccee",lty=2)
# remember to redraw the points
points(disptest.means[row.order])
```

`dotplot.mtb` *Minitab style dotplots.*

Description

Create a dotplot of a data vector in the sense of “dotplot” as used in the Minitab© package.

Usage

```
dotplot.mtb(x, xlim = NULL, main = NULL, xlab = NULL, ylab = NULL,
            pch = 19, hist = FALSE, yaxis = FALSE, mtbstyle=TRUE)
```

Arguments

`x`	A numeric vector.
`xlim`	The x limits of the plot.
`main`	A title for the plot; defaults to blank.
`xlab`	A label for the x axis; defaults to blank.
`ylab`	A label for the y axis; defaults to blank.
`pch`	The plotting symbol for the dots in the plot; defaults to a solid disc.
`hist`	Logical scalar; should the plot be done “histogram style, i.e. using vertical lines rather than stacks of dots?
`yaxis`	Logical scalar; should a y-axis be produced?
`mtbstyle`	Logical scalar; should the dotplot be done in the “Minitab” style? I.e. should the zero level be at the vertical midway point?

Details

The result of `hist=TRUE` looks less ugly than stacks of dots for very large data sets.

Value

Nothing. A plot is produced as a side effect.

Warnings

This function does something toadally different from the `dotplot()` (now `dotchart()`) function in the graphics package.

The labelling of the `y`-axis is device dependent.

Author(s)

Barry Rowlingson ⟨B.Rowlingson@lancaster.ac.uk⟩ and Rolf Turner ⟨r.turner@auckland.ac.nz⟩

Examples

```
## Not run:
set.seed(42)
x <- rpois(100,10)
dotplot.mtb(x,main="No y-axis.")
dotplot.mtb(x,yaxis=TRUE,main="With y-axis displayed.")
dotplot.mtb(x,hist=TRUE,main="An \"h\" style plot.")
dotplot.mtb(x,xlim=c(4,16),main="With the x-axis limited.")
dotplot.mtb(x,yaxis=TRUE,mtbstyle=FALSE,main="Non-Minitab style.")
dotplot.mtb(x,yaxis=TRUE,xlab="x",ylab="count",
            main="With x and y axis labels.")
## End(Not run)
```

`draw.arc` *Draw arc*

Description

Draw one or more arcs using classic graphics.

Usage

```
 draw.arc(x=1,y=NULL,radius=1,angle1=deg1*pi/180,angle2=deg2*pi/180,
  deg1=0,deg2=45,n=35,col=1,...)
```

Arguments

`x`	x coordinate of center. Scalar or vector.
`y`	y coordinate of center. Scalar or vector.
`radius`	radius. Scalar or vector.
`angle1`	Starting angle in radians. Scalar or vector.
`angle2`	Ending angle in radians. Scalar or vector.
`deg1`	Starting angle in degrees. Scalar or vector.
`deg2`	Ending angle in degrees. Scalar or vector.
`n`	Number of polygons to use to approximate the arc.
`col`	Arc colors.
`...`	Other arguments passed to segments. Vectorization is not supported for these.

Details

Draws one or more arcs from `angle1` to `angle2`. If `angle1` is numerically greater than `angle2`, then the angles are swapped.

Be sure to use an aspect ratio of 1 as shown in the example to avoid distortion.

Value

Returns a matrix of expanded arguments invisibly.

Author(s)

Gabor Grothendieck

Examples

```
plot(1:10, asp = 1,main="Test draw.arc")
draw.arc(5, 5, 1:10/10, deg2 = 1:10*10, col = "blue")
draw.arc(8, 8, 1:10/10, deg2 = 1:10*10, col = 1:10)

# example taken from post by Hans Borcher:
# https://stat.ethz.ch/pipermail/r-help/2009-July/205728.html
# Note setting of aspect ratio to 1 first.
curve(sin(x), 0, pi, col="blue", asp=1)
draw.arc(pi/2, 0, 1, deg1=45, deg2=135, col="red")
```

`draw.circle` *Draw a circle.*

Description

Draws a circle on an existing plot.

Usage

```
draw.circle(x,y,radius,nv=100,border=NULL,col=NA,lty=1,lwd=1)
```

Arguments

`x,y`	Coordinates of the center of the circle.
`radius`	Radius of the circle in user units.
`nv`	Number of vertices to draw the circle.
`border`	Color to use for drawing the circumference.
`col`	Color to use for filling the circle.
`lty`	Line type for the circumference.
`lwd`	Line width for the circumference.

Value

A list with the x and y coordinates of the points on the circumference.

Note

The principal advantage of '`draw.circle`' is that it adjusts for the aspect ratio of the plot.

Author(s)

Jim Lemon

See Also

'`polygon`'

Examples

```
plot(1:5,seq(1,10,length=5),type="n",xlab="",ylab="",main="Test draw.circle")
draw.circle(2,2,0.5,border="purple",lty=1,lwd=1)
draw.circle(2.5,8,0.6,border="red",lty=3,lwd=3)
draw.circle(4,3,0.7,border="green",lty=1,lwd=1)
draw.circle(3.5,7,0.8,border="blue",lty=2,lwd=2)
```

`draw.tilted.sector` *Display a 3D pie sector*

Description

Displays a 3D pie sector.

Usage

```
draw.tilted.sector(x=0,y=0,edges=100,radius=1,height=0.3,theta=pi/6,
 start=0,end=pi*2,border=par("fg"),col=par("bg"),explode=0,shade=0.8)
```

Arguments

`x,y`	Position of the center of the pie sector in user units
`edges`	Number of edges to draw a complete ellipse
`radius`	the radius of the pie in user units
`height`	the height of the pie in user units
`theta`	The angle of viewing in radians
`start`	Starting angle of the sector
`end`	Ending angle of the sector
`border`	The color of the sector border lines
`col`	Color of the sector
`explode`	How far to "explode" the sectors in user units
`shade`	If > 0 and < 1, the proportion to reduce the brightness of the sector color to get a better 3D effect.

Details

'`draw.tilted.sector`' displays a single 3D pie sector. It is probably only useful when called from '`pie3D`'. The '`shade`' argument proportionately reduces the brightness of the RGB color of the sector to produce a top lighted effect.

Value

The bisector of the pie sector in radians.

Author(s)

Jim Lemon

See Also

'`pie3D`'

`emptyspace` *Find an empty space on a plot.*

Description

Try to find the largest empty rectangle on a plot.

Usage

```
emptyspace(x,y=NA,bars=FALSE)
```

Arguments

`x,y`	x and y positions of the points or centers and heights of the bars
`bars`	Whether to add x and y points to represent the bars if the plot is a barplot.

Details

'`emptyspace`' divides the area defined by '`par("usr")`' into smaller and smaller rectangles until at least one rectangle has no points defined by '`x`' and '`y`' within it. It then tries to find the largest such rectangle if more than one exists and calculates its center. If the plot is very crowded, the resulting rectangle may be very small.

'`emptyspace`' will accept a list of at least two matrices as if it was returned from '`brkdn.plot`', calculate the positions of the ends of the dispersion bars and then try to find an empty rectangle. It will also accept a list of x-y coordinates, looking for the first element to have the name '`x`', and set '`y`' to the second element.

Note that if there are any NAs in '`x`' or '`y`', emptyspace will fail.

Value

The ‘x’ and ‘y’ coordinates of the center of the rectangle found.

Author(s)

Jim Lemon

Examples

```
x<-rnorm(10)
y<-rnorm(10)
plot(x,y,main="Find the empty space",xlab="X",ylab="Y")
es<-emptyspace(x,y)
boxed.labels(es,labels="Here is the\nempty space")
```

---

fan.plot *Display a fan plot.*

Description

Displays numerical values as the arcs of overlapping sectors.

Usage

```
fan.plot(x,edges=200,radius=1,col=NULL,align.at=NULL,max.span=NULL,
 labels=NULL,labelpos=NULL,label.radius=1.2,align="left",shrink=0.02,
 main="",ticks=NULL,include.sumx=FALSE,...)
```

Arguments

x	Vector of numbers.
edges	The number of edges with which to draw a circle.
radius	The radius of the sectors.
col	The colors with which to fill the sectors.
align.at	Where to align the sectors (see Details).
max.span	The angle of the maximal sector in radians. The default is to scale ‘x’ so that it sums to 2*pi.
labels	Labels placed around the sector arcs.
labelpos	Optional circumferential positions for the labels.
label.radius	How far away from the sectors the labels will be placed. May be a vector with a radius for each label.
align	Position of the alignment of sectors (see Details).
shrink	How much to shrink each successive sector in user units.
main	Optional title for the plot.

`ticks`	The number of ticks that would appear if the sectors were on a pie chart. Default is no ticks, TRUE gives the number of ticks equal to the integer sum of ‘`x`’, which is fairly sensible if ‘`x`’ is a vector of integers.
`include.sumx`	Whether to include the sum of all ‘`x`’ values as the largest sector.
`...`	Additional arguments passed to ‘`polygon`’.

Details

The fan plot is a variant of the pie chart that places the sectors "on top" of each other from the largest to the smallest. By default, the largest sector is centered with its circumferential arc upwards, giving the plot the appearance of a folding fan. Passing a value for ‘`align.at`’ will place the point of alignment at that angle in radians. The sectors may be aligned at either the left or right edges or in the center. Note that ‘`align`’ must be one of ‘`left right`’ or ‘`center`’. Each successive sector is radially "shrunk" by a constant amount so that two equal sectors will both be visible.

In cases where there are several segments with very small differences, the labels may be crowded. There is a simple routine in the function to spread out crowded labels, but it may not be sufficient for severe crowding. By capturing the return value and manually altering the label positions, the crowded labels can be separated. This new vector of positions may then be passed as ‘`labelpos`’.

The calculation of sectors tries to ensure that they are circular. Thus there will be white space if the plotting device dimensions are not proportional to the size of the plot. The user must adjust the dimensions of the plotting device to get the correct appearance.

Value

The circumferential positions of the labels in radians. These are returned in order of decreasing size of the values plotted.

Author(s)

Jim Lemon, Anupam Tyagi

See Also

‘`floating.pie`’

Examples

```
# IUCN counts of threatened species by geographical area
iucn.df<-data.frame(area=c("Africa","Asia","Europe","N&C America",
 "S America","Oceania"),threatened=c(5994,7737,1987,4716,5097,2093))
fan.plot(iucn.df$threatened,
 labels=paste(iucn.df$area,iucn.df$threatened,sep="-"),
 main="Threatened species by geographical area",ticks=276)
# expand the plot to a semicircle
fan.plot(iucn.df$threatened,max.span=pi,
 labels=paste(iucn.df$area,iucn.df$threatened,sep="-"),
 main="Threatened species by geographical area",ticks=276)
# expand further to 3/4 of a circle
fan.plot(iucn.df$threatened,max.span=1.5*pi,
 labels=paste(iucn.df$area,iucn.df$threatened,sep="-"),
 main="Threatened species by geographical area",ticks=276)
```

`feather.plot` *Display vectors along a horizontal reference line.*

Description

Displays vectors along a line usually representing time or position.

Usage

```
feather.plot(r,theta,xpos,yref=0,use.arrows=TRUE,
col.refline="lightgray",fp.type="s",main="",xlab="",ylab="",
xlabels=NULL,...)
```

Arguments

`r`	radii of vectors
`theta`	direction of vectors in radians
`xpos`	where to start each vector along the reference line
`yref`	vertical position to place the reference line
`use.arrows`	whether to put arrow heads on the ends of the vectors
`col.refline`	the color of the reference line
`fp.type`	whether to use "standard" coordinates (begin at the right and move counterclockwise) or "meteorological" coordinates (begin at the top and move clockwise) when interpreting the values of '`theta`'
`main`	the title of the plot
`xlab`	the label for the reference line
`ylab`	the label for the vertical axis
`xlabels`	optional labels for the reference line
`...`	additional arguments passed to '`arrows`' or '`segments`'

Details

This function places vectors of length '`r`' and angle '`theta`' along a reference line that may represent time or position or some other value. The user is responsible for spacing the vectors so that they do not overlap if this is desired.

Value

nil

Author(s)

Jim Lemon, Eduardo Klein

See Also

‘`spread.labels`’

Examples

```
feather.plot(0.6+rnorm(8)/5,seq(0,7*pi/4,by=pi/4),1:8,
 main="Standard Coordinates",xlab="Time",ylab="Value")
if(dev.interactive()) par(ask=TRUE)
feather.plot(0.6+rnorm(8)/5,seq(0,7*pi/4,by=pi/4),1:8,
 main="Meteorological Coordinates",xlab="Time",ylab="Value",
 fp.type="m",xlabels=TRUE)
par(ask=FALSE)
```

`floating.pie` *Display a floating pie chart*

Description

Displays a pie chart at an arbitrary position on an existing plot

Usage

```
floating.pie(xpos,ypos,x,edges=200,radius=1,col=NULL,startpos=0,
 shadow=FALSE,...)
```

Arguments

`xpos,ypos`	x and y position of the center of the pie chart
`x`	a numeric vector for which each value will be a sector
`edges`	the number of lines forming a circle
`radius`	the radius of the pie in user units
`col`	the colors of the sectors - defaults to ‘`rainbow`’
`startpos`	The starting position for drawing sectors in radians.
`shadow`	Logical - whether to draw a shadow
`...`	graphical parameters passed to ‘`polygon`’

Value

The bisecting angle of the sectors in radians. Useful for placing text labels for each sector.

Note

As with most pie charts, simplicity is essential. Trying to display a complicated breakdown of data rarely succeeds.

Author(s)

Jim Lemon

See Also

'`pie.labels`', '`boxed.labels`', '`polygon.shadow`'

Examples

```
plot(1:5,type="n",main="Floating Pie test",xlab="",ylab="",axes=FALSE)
box()
polygon(c(0,0,5.5,5.5),c(0,3,3,0),border="#44aaff",col="#44aaff")
floating.pie(1.7,3,c(2,4,4,2,8),radius=0.5,
 col=c("#ff0000","#80ff00","#00ffff","#44bbff","#8000ff"))
floating.pie(3.1,3,c(1,4,5,2,8),radius=0.5,
 col=c("#ff0000","#80ff00","#00ffff","#44bbff","#8000ff"))
floating.pie(4,1.5,c(3,4,6,7),radius=0.5,
 col=c("#ff0066","#00cc88","#44bbff","#8000ff"))
draw.circle(3.9,2.1,radius=0.04,col="white")
draw.circle(3.9,2.1,radius=0.04,col="white")
draw.circle(3.9,2.1,radius=0.04,col="white")
draw.circle(4,2.3,radius=0.04,col="white")
draw.circle(4.07,2.55,radius=0.04,col="white")
draw.circle(4.03,2.85,radius=0.04,col="white")
text(c(1.7,3.1,4),c(3.7,3.7,3.7),c("Pass","Pass","Fail"))
```

`fullaxis` *Add an axis with a line to the edge of the plot*

Description

As '`axis`', but draws a "box" line in the same color as the axis.

Usage

```
fullaxis(side=1,at=NULL,labels=TRUE,line=NA,pos=NA,outer=FALSE,
font=NA,lty="solid",lwd=1,lwd.ticks=lwd,col=NULL,col.ticks=NULL,
hadj=NA,padj=NA,...)
```

Arguments

`side`	The side of the plot to draw the axis
`at`	Optional positions in user units for the tick marks.
`labels`	Optional labels for the tick marks.
`line`	Optional line into the margin.
`pos`	Optional position in user units for the axis. Defaults to the edge.
`outer`	Whether to use the outer margin as for '`axis`'.
`font`	Font for the labels.
`lty`	Line type.
`lwd`	Line width for the axis.
`lwd.ticks`	Line width for the ticks.
`col`	color for the axis and tick marks. See Details for label color.
`col.ticks`	Color for the tick marks if different from the axis.
`hadj,padj`	Justification for the labels. See '`axis`'.
`...`	Further arguments passed to '`axis`'.

Details

'`fullaxis`' draws a line to the edges of the plot and then calls '`axis`' to draw an axis. '`fullaxis`' is mainly useful for drawing a colored axis on a boxed plot. In order to get the tick labels the same color as the axis and ticks, pass the '`col.axis`' argument as well as '`col`'. See the example for some useful tips.

Value

The positions of the tick marks in user units.

Author(s)

Jim Lemon

See Also

'`axis`'

Examples

```
plot(runif(20,-1,1),runif(20,-1,1),xlim=c(-1,1.5),main="Demo of fullaxis",
 xlab="X",ylab="Y",axes=FALSE)
fullaxis(1,col="red")
fullaxis(2,col="blue",col.axis="blue")
fullaxis(4,at=c(-0.5,0,0.5),labels=c("Negative","Zero","Positive"),pos=1.2,
 col="green",las=1)
# add a top line to complete the "box"
xylim<-par("usr")
segments(xylim[1],xylim[4],xylim[2],xylim[4])
```

furc *Plot a dendrite.*

Description

Plot one level of a dendrogram displaying two or more mutually exclusive attributes.

Usage

```
furc(x,xpos,yrange,toplevel,cex=1)
```

Arguments

`x`	A ‘`dendrite`’ object containing the counts of objects having combinations of mutually exclusive attributes.
`xpos`	The horizontal position on the plot to display the current level of the dendrogram.
`yrange`	The range of values in which the current level of the dendrogram will be displayed.
`toplevel`	A flag for the function to know whether it is at the top level of the dendrogram or not. Do not change this argument.
`cex`	The character expansion to use in the display.

Details

‘`furc`’ displays one *furc*ation of the dendrogram. A furcation is a single box displaying its label and count that may split into finer divisions. If so, ‘`furc`’ calls itself for each furcation until there are no more splits.

Value

nil

Author(s)

Jim Lemon

See Also

‘`plot.dendrite`’

gantt.chart *Display a Gantt chart*

Description

Displays a Gantt chart with priority coloring

Usage

```
gantt.chart(x=NULL,format="%Y/%m/%d",xlim=NULL,taskcolors=NULL,
 priority.legend=FALSE,vgridpos=NULL,vgridlab=NULL,vgrid.format="%Y/%m/%d",
 half.height=0.25,hgrid=FALSE,main="",xlab="",cylindrical=FALSE)
```

Arguments

`x`	a list of task labels, start/end times and task priorities as returned by '`get.gantt.info`'. If this is not present, '`get.gantt.info`' will be called.
`format`	the format to be used in entering dates/times (see '`strptime`').
`xlim`	the horizontal limits of the plot.
`taskcolors`	a vector of colors used to illustrate task priority.
`priority.legend`	Whether to display a priority color legend.
`vgridpos`	optional positions of the vertical grid lines.
`vgridlab`	optional labels for the vertical grid lines.
`vgrid.format`	format for the vertical grid labels.
`half.height`	the proportion of the spacing between task bars that will be filled by the bar on each side - 0.5 will leave no space.
`hgrid`	logical - whether to display grid lines between the bars.
`main`	the title of the plot - note that this is actually displayed using '`mtext`'.
`xlab`	horizontal axis label - usually suppressed.
`cylindrical`	Whether to give the bars a cylindrical appearance.

Details

If task priority colors are not wanted, set '`taskcolors`' to a single value to suppress the coloring. If this is not done, '`rainbow`' will be called to generate a different color for each task.

There can now be more than one time interval for each task. If there is, more than one bar will be displayed for each "task", which may not be a task at all, but intervals of some attribute for each entity. See the last example.

Value

The list used to create the chart - see '`get.gantt.info`' for details. This can be saved and reused rather than manually entering the information each time the chart is displayed.

Author(s)

Jim Lemon (original by Scott Waichler - features by Ulrike Gromping)

See Also

'get.gantt.info'

Examples

```
Ymd.format<-"%Y/%m/%d"
gantt.info<-list(labels=
 c("First task","Second task","Third task","Fourth task","Fifth task"),
 starts=
 as.POSIXct(strptime(
 c("2004/01/01","2004/02/02","2004/03/03","2004/05/05","2004/09/09"),
 format=Ymd.format)),
 ends=
 as.POSIXct(strptime(
 c("2004/03/03","2004/05/05","2004/05/05","2004/08/08","2004/12/12"),
 format=Ymd.format)),
 priorities=c(1,2,3,4,5))
vgridpos<-as.POSIXct(strptime(c("2004/01/01","2004/02/01","2004/03/01",
 "2004/04/01","2004/05/01","2004/06/01","2004/07/01","2004/08/01",
 "2004/09/01","2004/10/01","2004/11/01","2004/12/01"),format=Ymd.format))
vgridlab<-
 c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec")
gantt.chart(gantt.info,main="Calendar date Gantt chart (2004)",
 priority.legend=TRUE,vgridpos=vgridpos,vgridlab=vgridlab,hgrid=TRUE)
# if both vgidpos and vgridlab are specified,
# starts and ends don't have to be dates
info2<-list(labels=c("Jim","Joe","Jim","John","John","Jake","Joe","Jed","Jake"),
 starts=c(8.1,8.7,13.0,9.1,11.6,9.0,13.6,9.3,14.2),
 ends=c(12.5,12.7,16.5,10.3,15.6,11.7,18.1,18.2,19.0))
gantt.chart(info2,vgridlab=8:19,vgridpos=8:19,
 main="Sitting at desk",taskcolors="lightgray")
```

---

gap.barplot *Display a barplot with a gap (missing range) on one axis*

Description

Displays a barplot with a missing range.

Usage

```
gap.barplot(y,gap,xaxlab,xtics,yaxlab,ytics,ylim=NA,xlab=NULL,
 ylab=NULL,horiz=FALSE,col,...)
```

Arguments

`y`	data values
`gap`	the range of values to be left out
`xaxlab`	labels for the x axis ticks
`xtics`	position of the x axis ticks
`yaxlab`	labels for the y axis ticks
`ytics`	position of the y axis ticks
`ylim`	optional y limits for the plot
`xlab`	label for the x axis
`ylab`	label for the y axis
`horiz`	whether to have vertical or horizontal bars
`col`	color(s) in which to plot the values
`...`	arguments passed to '`barplot`'.

Details

Displays a barplot omitting a range of values on the X or Y axis. Typically used when there is a relatively large gap in the range of values represented as bar heights. See '`axis.break`' for a brief discussion of plotting on discontinuous coordinates.

If the user does not ask for specific y limits, the function will calculate limits based on the range of the data values. If passing specific limits, remember to subtract the gap from the upper limit.

Value

The center positions of the bars.

Author(s)

Jim Lemon

See Also

'`gap.barplot`'

Examples

```
twogrp<-c(rnorm(10)+4,rnorm(10)+20)
gap.barplot(twogrp,gap=c(8,16),xlab="Index",ytics=c(3,6,17,20),
 ylab="Group values",main="Barplot with gap")
gap.barplot(twogrp,gap=c(8,16),xlab="Index",ytics=c(3,6,17,20),
 ylab="Group values",horiz=TRUE,main="Horizontal barplot with gap")
```

gap.boxplot *Display a boxplot with a gap (missing range)*

Description

Displays a boxplot with a missing range.

Usage

```
gap.boxplot(x,...,gap=list(top=c(NA,NA),bottom=c(NA,NA)),
range=1.5,width=NULL,varwidth=FALSE,notch=FALSE,outline=TRUE,
names,ylim=NA,plot=TRUE,border=par("fg"),col=NULL,log="",
axis.labels=NULL,pars=list(boxwex=0.8,staplewex=0.5,outwex=0.5),
horizontal=FALSE,add=FALSE,at=NULL,main=NULL)
```

Arguments

`x`	numeric vector or a list of vectors
`...`	arguments passed to ‘`boxplot`’.
`gap`	the range(s) to be omitted - a list with two components, ‘`top`’ and ‘`bottom`’ each specifying a range to omit. The default range of ‘`c(NA,NA)`’ means no omitted range
`range`	how far to extend the whiskers, (see ‘`boxplot`’)
`width`	the relative widths of the boxes
`varwidth`	if TRUE, box widths are proportional to the square roots of the number of observations
`notch`	whether to display the confidence intervals for the median as notches
`outline`	whether to display outliers
`names`	optional names to display beneath each boxplot
`ylim`	Optional y axis limits for the plot.
`boxwex`	scale factor for box widths
`staplewex`	staple width proportional to box width
`outwex`	outlier line width
`plot`	dummy argument for consistency with ‘`boxplot`’ - always plots
`border`	optional color(s) for the box lines
`col`	optional color(s) to fill the boxes
`log`	whether to use a log scale - currently does nothing
`axis.labels`	Optional axis labels.
`pars`	optional parameters for consistency with ‘`boxplot`’
`horizontal`	whether to plot horizontal boxplots - currently does nothing
`add`	whether to add the boxplot(s) to a current plot - currently does nothing
`at`	optional horizontal locations for the boxplots - currently does nothing
`main`	a title for the plot

Details

Displays boxplot(s) omitting range(s) of values on the top and/or bottom of the plot. Typically used when there are outliers far from the boxes. See '`boxplot`' for more detailed descriptions of the arguments. If the gaps specified include any of the values in the '`stats`' matrix returned from '`boxplot`', the function will exit with an error message. This prevents generation of NAs in indexing operations, which would fail anyway. A gap can include part of a box, but it is unlikely that this would be intended by the user.

See '`axis.break`' for a brief discussion of plotting on discontinuous coordinates.

Value

A list with the same structure as returned by '`boxplot`', except that the values of elements beyond the gap(s) have their true positions on the plot rather than the original values.

Author(s)

Jim Lemon

See Also

'`gap.barplot`','`gap.plot`'

Examples

```
twovec<-list(vec1=c(rnorm(30),-6),vec2=c(sample(1:10,40,TRUE),20))
gap.boxplot(twovec,gap=list(top=c(12,18),bottom=c(-5,-3)),
main="Show outliers separately")
if(dev.interactive()) par(ask=TRUE)
gap.boxplot(twovec,gap=list(top=c(12,18),bottom=c(-5,-3)),range=0,
main="Include outliers in whiskers")
par(ask=FALSE)
```

`gap.plot` *Display a plot with one or two gaps (missing ranges) on one axis*

Description

Displays a plot with one or two missing ranges on one of the axes.

Usage

```
gap.plot(x,y,gap,gap.axis="y",bgcol="white",breakcol="black",brw=0.02,
 xlim,ylim,xticlab,xtics=NA,yticlab,ytics=NA,lty=rep(1,length(x)),
 col=rep(par("col"),length(x)),pch=rep(1,length(x)),add=FALSE,...)
```

Arguments

`x,y`	data values
`gap`	the range(s) of values to be left out
`gap.axis`	whether the gaps are to be on the x or y axis
`bgcol`	the color of the plot background
`breakcol`	the color of the "break" marker
`brw`	break width relative to plot width
`xlim,ylim`	the plot limits.
`xticlab`	labels for the x axis ticks
`xtics`	position of the x axis ticks
`yticlab`	labels for the y axis ticks
`ytics`	position of the y axis ticks
`lty`	line type(s) to use if there are lines
`col`	color(s) in which to plot the values
`pch`	symbols to use in plotting.
`add`	whether to add values to an existing plot.
`...`	arguments passed to '`plot`' and '`points`'.

Details

Displays a plot omitting one or two ranges of values on one axis. Typically used when there is a relatively large gap or two in the overall range of one set of values, often because of outliers. The function warns the user if any values have been omitted by being in the "gap". See '`axis.break`' for a brief discussion of plotting on discontinuous coordinates.

To add more data series to a gap plot, call '`gap.plot`' with '`add = TRUE`'. The same '`gap`' and '`gap.axis`' arguments as in the initial call must be passed or the data will not be displayed correctly. Remember to pass an explicit '`xlim`' or '`ylim`' to the initial call if the added data exceed the range of the data initially displayed. Also remember to subtract the width(s) of the gap(s) if you are passing an explicit '`xlim`' or '`ylim`'.

Because the gaps take up some space, it is possible to have a data value that is just below a gap plotted in the gap. The answer is to make the lower gap limit a little higher if this is a problem.

If at least four values are passed in '`gap`', the first four will be used to calculate two "gaps" in the plot instead of one. The function does not check whether these values are sensible, so it is quite easy to ask for a very silly plot.

The default ticks are usually not ideal, and most users will want to pass their own tick positions and perhaps labels. Note that '`lines`' appears to use only the first '`col`' and '`lty`' argument value, so if you must have lines with different colors and types, use '`add=TRUE`' and add them separately (see the third example for the problem and the solution).

Value

nil

Author(s)

Jim Lemon and Ben Bolker (thanks to Zheng Lu for the "add" idea, and Art Roberts for helping to get the gaps right.)

See Also

'gap.barplot', 'axis.break'

Examples

```
twogrp<-c(rnorm(5)+4,rnorm(5)+20,rnorm(5)+5,rnorm(5)+22)
gpcol<-c(2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5)
gap.plot(twogrp,gap=c(8,16),xlab="Index",ylab="Group values",
 main="Gap on Y axis",col=gpcol)
gap.plot(twogrp,rnorm(20),gap=c(8,16),gap.axis="x",xlab="X values",
 xtics=c(4,7,17,20),ylab="Y values",main="Gap on X axis with added lines")
gap.plot(c(seq(3.5,7.5,by=0.5),seq(16.5,22.5,by=0.5)),
 rnorm(22),gap=c(8,16),gap.axis="x",type="l",add=TRUE,col=2,)
gap.plot(twogrp,gap=c(8,16,25,35),
xlab="X values",ylab="Y values",xlim=c(1,30),ylim=c(0,25),
main="Test two gap plot with the lot",xtics=seq(0,30,by=5),
ytics=c(4,6,18,20,22,38,40,42),
lty=c(rep(1,10),rep(2,10)),
pch=c(rep(2,10),rep(3,10)),
col=c(rep(2,10),rep(3,10)),
type="b")
gap.plot(21:30,rnorm(10)+40,gap=c(8,16,25,35),add=TRUE,
 lty=rep(3,10),col=rep(4,10),type="l")
```

`get.breaks` *Get the breakpoints for a weighted histogram*

Description

Gets the breakpoints for a weighted histogram.

Usage

```
get.breaks(x,breaks)
```

Arguments

x	A numeric vector.
breaks	Either the name of the function to calculate breakpoints, the number of categories or a vector of breakpoints.

Details

'`get.breaks`' either calls the same functions as '`hist`' to get breakpoints or calculates a given number or just returns '`breaks`' if they are already specified.

Value

A vector of breakpoints.

Author(s)

Jim Lemon

See Also

'`hist`'

`get.gantt.info` *Gather the information to create a Gantt chart*

Description

Allows the user to enter the information for a Gantt chart.

Usage

```
get.gantt.info(format="%Y/%m/%d")
```

Arguments

`format`	the format to be used in entering dates/times. Defaults to YYYY/mm/dd. See '`strptime`' for various date/time formats.

Value

The list used to create the chart. Elements are:

`labels`	The task labels to be displayed at the left of the chart.
`starts,ends`	The task starts/ends as POSIXct dates/times.
`priorities`	Task priorities as integers in the range 1 to 10. There can be less than 10 levels of priority, but if priorities do not start at 1 (assumed to be the highest), the default priority colors will be calculated from 1.

Author(s)

Jim Lemon

See Also

'gantt.chart'

Examples

```
cat("Enter task times using HH:MM (hour:minute) format\n")
get.gantt.info("%H:%M")
```

---

get.segs *Calculate the midpoints and limits for a centipede plot*

Description

Calculates midpoints and limits for a list or data frame for use with centipede.plot.

Usage

```
get.segs(x,mct="mean",lower.limit="std.error",upper.limit=lower.limit)
```

Arguments

`x`	a list or data frame.
`mct`	The name of the function to calculate midpoints.
`lower.limit,upper.limit`	The names of the function(s) to calculate lower and upper limits.

Details

'get.segs' calls the functions whose names are passed to calculate midpoints and limits for each list element or data frame column. The user can define special functions for the central and dispersion measures if desired.

Value

A matrix with four rows and as many columns as were in the object 'x'. The first row contains the midpoint values, the second and third the lower and upper limit values respectively and the fourth row the number of valid observations in the columns.

Author(s)

Jim Lemon

See Also

'centipede.plot'

`get.soil.texture` *Enter soil texture data.*

Description

'`get.soil.texture`' calls '`get.triprop`' to allow the user to enter soil textures as the proportions or percentages of three components, sand, silt and clay.

Usage

```
get.soil.texture(use.percentages=FALSE,cnames=c("sand","silt","clay"))
```

Arguments

`use.percentages`	Logical - whether to treat the entries as percentages and scale to proportions.
`cnames`	column names for the resulting three column matrix.

Value

A matrix of the components of one or more soil samples.

Author(s)

Sander Oom and Jim Lemon

See Also

'`soil.texture`','`get.triprop`'

Examples

```
if(dev.interactive()) {
 newsp<-get.soil.texture()
 # show the soil triangle
 soil.texture()
 # now plot the observations
 show.soil.texture(newsp)
}
```

get.triprop — *Enter three proportion data - usually soil textures.*

Description

'get.triprop' allows the user to enter triplets of proportions or percentages of three components such as sand, silt and clay in soils.

Usage

```
get.triprop(use.percentages=FALSE,cnames=c("1st","2nd","3rd"))
```

Arguments

use.percentages	Logical - whether to treat the entries as percentages and scale to proportions.
cnames	column names for the resulting three column matrix.

Details

The three proportions of each row must sum to 100 or 1 within 1% or the function will warn the operator.

Value

A matrix of the components of one or more observations.

Author(s)

Jim Lemon

See Also

'triax.plot', 'soil.texture'

Examples

```
if(dev.interactive()) {
 # get some proportions
 newsp<-get.triprop()
 # show the triangle
 triax.frame(main="Test triax.plot")
 # now plot the observations
 triax.points(newsp)
}
```

`getFigCtr` *Find the center of the figure region.*

Description

Calculates the center of the figure region.

Usage

```
getFigCtr()
```

Details

'`getFigCtr`' reads parameters about the current plot and calculates the vertical and horizontal centers of the figure region. This is typically useful for placing a centered title on plots where the left and right margins are very different.

Value

A two element vector containing the coordinates of the center of the figure region in user units.

Author(s)

Jim Lemon

`getIntersectList` *Enter the information for a set intersection display*

Description

Enter the information for a set intersection display.

Usage

```
getIntersectList(nelem,xnames=NULL,sep="-")
```

Arguments

`nelem`	The number of sets for which the intersections will be displayed.
`xnames`	The labels for the set intersections. The function creates names from combinations of the first '`nelem`' capital letters if none are given.
`sep`	The separator to use when calling '`paste`'.

Details

'getIntersectList' allows the user to manually enter the counts of set intersections rather than build this information from a matrix of data. It is probably most useful for producing an intersection diagram when the counts of the intersections are already known, or when the values are proportions rather than counts as in the example.

Value

A list of the counts of elements in the set intersections.

Author(s)

Jim Lemon

See Also

'makeIntersectList', 'intersectDiagram'

Examples

```
 # this example is from a haplotype mapping problem submitted by Mao Jianfeng
 ## Not run:
 hapIntList<-
  getIntersectList(3,xnames=c("hap.Pd","hap.Pt","hap.Py"))
 # enter the data as follows:
 # Number of elements in hap.Pd - 1: 27.586
 # Number of elements in hap.Pt - 1: 20.689
 # Number of elements in hap.Py - 1: 31.035
 # Number of elements in hap.Pd-hap.Pt - 1: 10.345
 # Number of elements in hap.Pd-hap.Py - 1: 10.345
 # Number of elements in hap.Pt-hap.Py - 1: 0
 # Number of elements in hap.Pd-hap.Pt-hap.Py - 1: 0
 # Total number of elements - 1: 100

## End(Not run)
 hapIntList<-structure(list(structure(c(27.586, 20.689, 31.035),
  .Names = c("hap.Pd","hap.Pt","hap.Py")),
  structure(c(10.345, 10.345, 0),
  .Names = c("hap.Pd-hap.Pt","hap.Pd-hap.Py","hap.Pt-hap.Py")),
  structure(0, .Names = "hap.Pd-hap.Pt-hap.Py"),100),
  class = "intersectList")
 intersectDiagram(hapIntList)
```

---

getMarginWidth *Find the margin width necessary to fit text or a legend next to a plot.*

Description

Calculates the margin width necessary to fit text or a legend next to a plot.

Usage

```
getMarginWidth(side=4,labels,is.legend=FALSE)
```

Arguments

`side`	Which side of the plot (as in axis).
`labels`	The text to place next to the plot.
`is.legend`	Whether the text is in a legend or not.

Details

'`getMarginWidth`' reads parameters about the current plot and calculates the left or right (default) margin necessary to fit the strings passed as '`labels`' or a legend containing those strings.

Value

A two element list containing the number of margin lines necessary to fit the text or legend and the horizontal center of the margin in user units.

Author(s)

Jim Lemon

Examples

```
plot(rnorm(10))
getMarginWidth(labels=c("Long label","Even longer label"))
```

`gradient.rect` *Display a rectangle filled with an arbitrary color gradient.*

Description

'`gradient.rect`' draws a rectangle consisting of '`nslices`' subrectangles of the colors in '`col`' or those returned by '`color.gradient`' if '`col`' is NULL. The rectangle is 'sliced' in the direction specified by '`gradient`'.

Usage

```
gradient.rect(xleft,ybottom,xright,ytop,reds,greens,blues,col=NULL,
 nslices=50,gradient="x",border=par("fg"))
```

Arguments

`xleft,ybottom,xright,ytop`	Positions of the relevant corners of the desired rectangle, as in '`rect`'.
`reds,greens,blues`	vectors of the values of the color components either as 0 to 1 or ,if any value is greater than 1, 0 to 255.
`col`	Vector of colors. If supplied, this takes precedence over '`reds,greens,blues`' and '`nslices`' will be set to its length.
`nslices`	The number of sub-rectangles that will be drawn.
`gradient`	whether the gradient should be horizontal (x) or vertical.
`border`	The color of the border around the rectangle (NA for none).

Value

the vector of hexadecimal color values from '`color.gradient`' or '`col`'.

Author(s)

Jim Lemon

Examples

```
# get an empty box
plot(0:10,type="n",axes=FALSE)
# run across the three primaries
gradient.rect(1,0,3,6,reds=c(1,0),
 greens=c(seq(0,1,length=10),seq(1,0,length=10)),
 blues=c(0,1),gradient="y")
# now a "danger gradient"
gradient.rect(4,0,6,6,c(seq(0,1,length=10),rep(1,10)),
 c(rep(1,10),seq(1,0,length=10)),c(0,0),gradient="y")
# now just a smooth gradient across the bar
gradient.rect(7,0,9,6,col=smoothColors("red",38,"blue"),border=NA)
```

`hexagon` *Draw a hexagon*

Description

Draws a hexagon on the current graphic device.

Usage

```
hexagon(x,y,unitcell=1,col=NA,border="black")
```

Arguments

`x,y`	x and y position of the bottom left corner of the square that would pack into the same space as the hexagon.
`unitcell`	The dimension of the side of the abovementioned square.
`col`	The color to fill the hexagon - default is no fill.
`border`	The color of the perimeter of the hexagon.

Value

nil

Note

Draws a hexagon with the same center as a square that would pack into the same dimensions as the hexagon. That is, given a grid of squares with alternate rows shifted one half the length of the sides, the hexagons drawn would be close packed. Its use in the plotrix package is to provide an alternative unit cell for the '`color2D.matplot`' function.

Author(s)

Jim Lemon

See Also

'`color2D.matplot`'

`hierobarp` *Display a nested breakdown of numeric values.*

Description

Breaks down a numeric element of a data frame by one or more categorical elements and displays the breakdown as a bar plot.

Usage

```
hierobarp(formula=NULL,data=NULL,maxlevels=10,mct=mean,lmd=std.error,umd=lmd,
x=NULL,xlim=NULL,ylim=NULL,main="",xlab="",ylab="",start=0,end=1,shrink=0.02,
errbars=FALSE,col=NA,labelcex=1,lineht=NA,showall=FALSE,barlabels=NULL,
showbrklab=TRUE,mar=NULL,arrow.cap=NA)
```

Arguments

`formula`	A formula with a numeric element of a data frame on the left and one or more categorical elements on the right.
`data`	A data frame containing the elements in '`formula`'.
`maxlevels`	The maximum number of levels in any categorical element. Mainly to prevent the mess caused by breaking down by a huge number of categories.
`mct`	The measure of central tendency function to use.
`lmd`	The lower measure of dispersion function to use.
`umd`	The upper measure of dispersion function to use.
`x`	This becomes the result of the breakdown after the first call. Currently anything passed as this argument will be ignored.
`xlim,ylim`	Optional x and y limits for the plot.
`main`	Title for the plot.
`xlab,ylab`	Axis labels for the plot.
`start,end`	The start and end values of the initial display. The user will almost certainly not want to change these.
`shrink`	The proportion to shrink the width of the bars as more levels are added. This proportion increases with the number of levels.
`errbars`	Whether to display error bars on the lowest level of breakdown.
`col`	The colors to use to fill the bars. See Details.
`barlabels`	Optional group labels that may be useful if the factors used to break down the numeric variable are fairly long strings.
`labelcex`	Character size for the group labels.
`lineht`	The height of a line of text in the lower margin of the plot in user units. This will be calculated by the function if a value is not passed.
`showall`	Whether to display bars for the entire breakdown.
`showbrklab`	Whether to display the labels for the lowest level of breakdown.
`mar`	If not NULL, a four element vector to set the plot margins. If new margins are set, the user must reset the margins after the function exits.
`arrow.cap`	The width of the "cap" on error bars in user units, defaulting to 0.01.

Details

'`hierobarp`' displays a bar plot illustrating the breakdown of a numeric element of a data frame by one or more categorical elements. The breakdown is performed by '`hierobrk`'. Typically, the mean of each category specified by the formula is displayed as the height of a bar. If '`showall`' is TRUE, the entire nested breakdown will be displayed. This can be useful in visualizing the relationship between groups and subgroups in a compact format.

The colors of the bars are determined by '`col`'. If '`showall`' is FALSE, the user only need pass a vector of colors, usually the same length as the number of categories in the final (first on the right side) element in the formula. If '`showall`' is TRUE and the user wants to color all of the bars, a list with as many elements as there are levels in the breakdown should be passed. Each element should be a vector of colors, again usually the same length as the number of categories. As the categorical variables are likely to be factors, it is important to remember that the colors must be in the correct order for the levels of the factors. When the levels are not in the default alphanumeric order, it is quite easy to get this wrong.

Value

The summary arrays produced by hierobrk.

Author(s)

Jim Lemon and Ofir Levy

See Also

'`hierobrk`', '`barp`'

Examples

```
 test.df<-data.frame(Age=rnorm(100,25,10),
  Sex=sample(c("M","F"),100,TRUE),
  Marital=sample(c("D","M","S","W"),100,TRUE),
  Employ=sample(c("Full Time","Part Time","Unemployed"),100,TRUE))
 test.col<-list(Overall="gray",Employ=c("#1affd8","#caeecc","#f7b3cc"),
  Marital=c("mediumpurple","orange","tan","lightgreen"),Sex=c("pink","lightblue"))
 hierobarp(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
  main="Show only the final breakdown",errbars=TRUE,col=test.col$Sex)
 # set up functions for 20 and 80 percentiles - must be offsets, not limits
 q20<-function(x,na.rm=TRUE) return(mean(x)-quantile(x,probs=0.2,na.rm=TRUE))
 q80<-function(x,na.rm=TRUE) return(quantile(x,probs=0.8,na.rm=TRUE)-mean(x))
 # show the asymmetric dispersion measures
 hierobarp(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
  main="Use median and quantiles for dispersion",mct=median,lmd=q20,umd=q80,
  errbars=TRUE,col=test.col$Sex)
 ## Not run:
  # start a wide plot window for this one
  x11(width=10)
  hierobarp(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
   main="Show the entire hierarchical breakdown",col=test.col,showall=TRUE,
   showbrklab=TRUE,mar=c(5,4,4,8))
  # example of a legend that might be included, needs a lot of space
  par(xpd=TRUE)
  legend(1.02,27,c("Overall","Full time","Part time","No work","Divorced",
   "Married","Single","Widowed","Female","Male"),
   fill=unlist(test.col))
  par(xpd=FALSE,mar=c(5,4,4,2))

## End(Not run)
```

`hierobarp.svymean` *Display a nested breakdown of means.*

Description

Displays a nested breakdown as a bar plot of means from the output of svyby (survey package) using svymean as the function.

Usage

```
hierobarp.svymean(x,meanvar,dispvar,xlim=NULL,ylim=NULL,
 main="",xlab="",ylab="",yticks=NULL,start=0,end=1,shrink=0.02,
 errbars=FALSE,col=NA,labelcex=1,lineht=NA,showall=TRUE,barlabels=NULL,
 showbrklab=TRUE,mar=NULL)
```

Arguments

`x`	The output of the '`svyby`' function using '`svymean`'. See Details.
`meanvar`	The name of the column containing the mean values.
`dispvar`	The name of the column containing the SE values.
`xlim,ylim`	Optional x and y limits for the plot.
`main`	Title for the plot.
`xlab,ylab`	Axis labels for the plot. X axis labels are usually left out
`yticks`	Optional tick labels for the y-axis.
`start,end`	The start and end values of the initial display. The user will almost certainly not want to change these.
`shrink`	The proportion to shrink the width of the bars as more levels are added. This proportion increases with the number of levels.
`errbars`	Whether to display error bars on the lowest level of breakdown.
`col`	The colors to use to fill the bars. See Details.
`labelcex`	Character size for the group labels.
`lineht`	The height of a line of text in the lower margin of the plot in user units. This will be calculated by the function if a value is not passed.
`showall`	Whether to display bars for the entire breakdown.
`barlabels`	Optional group labels that may be useful if the factors used to break down the numeric variable are fairly long strings.
`showbrklab`	Whether to display the labels for the lowest level of breakdown.
`mar`	If not NULL, a four element vector to set the plot margins. If new margins are set, the user must reset the margins after the function exits.

Details

'`hierobarp.svymean`' displays a bar plot illustrating the breakdown of a numeric variable as group means. This can be useful in visualizing the relationship between groups and subgroups in a compact format. While the function can be used with just the output of the '`svyby`' and '`svymean`' functions, the user will almost certainly have the **survey** package installed.

The colors of the bars are determined by '`col`'. A list with as many elements as there are levels in the breakdown should be passed. Each element should be a vector of colors, again usually the same length as the number of categories. As the categorical variables are likely to be factors, it is important to remember that the colors must be in the correct order for the levels of the factors. When the levels are not in the default alphanumeric order, it is quite easy to get this wrong.

Value

nil

Author(s)

Jim Lemon

See Also

'`hierobarp`', '`barp`'

`hierobarp.svyprop` *Display a nested breakdown of numeric proportions.*

Description

Displays a nested breakdown as a bar plot of proportions from the output of svyby (survey package) using svytotal as the function.

Usage

```
hierobarp.svyprop(x,truevar,falsevar,xlim=NULL,ylim=NULL,
 main="",xlab="",ylab="",yticks=NULL,start=0,end=1,shrink=0.02,
 errbars=FALSE,col=NA,labelcex=1,lineht=NA,showall=TRUE,barlabels=NULL,
 showbrklab=TRUE,mar=NULL)
```

Arguments

`x`	The output of the '`svyby`' function using '`svytotal`' when there are two outcomes. See Details.
`truevar`	The name of the column containing the count of TRUE values.
`falsevar`	The name of the column containing the count of FALSE values.
`xlim,ylim`	Optional x and y limits for the plot.
`main`	Title for the plot.

`xlab,ylab`	Axis labels for the plot.
`yticks`	Optional tick labels for the y-axis.
`start,end`	The start and end values of the initial display. The user will almost certainly not want to change these.
`shrink`	The proportion to shrink the width of the bars as more levels are added. This proportion increases with the number of levels.
`errbars`	Whether to display error bars on the lowest level of breakdown.
`col`	The colors to use to fill the bars. See Details.
`labelcex`	Character size for the group labels.
`lineht`	The height of a line of text in the lower margin of the plot in user units. This will be calculated by the function if a value is not passed.
`showall`	Whether to display bars for the entire breakdown.
`barlabels`	Optional group labels that may be useful if the factors used to break down the numeric variable are fairly long strings.
`showbrklab`	Whether to display the labels for the lowest level of breakdown.
`mar`	If not NULL, a four element vector to set the plot margins. If new margins are set, the user must reset the margins after the function exits.

Details

'`hierobarp.svyprop`' displays a bar plot illustrating the breakdown of a binary outcome as proportions. This can be useful in visualizing the relationship between groups and subgroups in a compact format. While the function can be used with just the output of the '`svyby`' and '`svytotal`' functions, the user will almost certainly have the **survey** package installed.

The colors of the bars are determined by '`col`'. A list with as many elements as there are levels in the breakdown should be passed. Each element should be a vector of colors, again usually the same length as the number of categories. As the categorical variables are likely to be factors, it is important to remember that the colors must be in the correct order for the levels of the factors. When the levels are not in the default alphanumeric order, it is quite easy to get this wrong.

Value

nil

Author(s)

Jim Lemon

See Also

'`hierobarp`', '`barp`'

hierobrk *Perform a nested breakdown of numeric values.*

Description

Breaks down a numeric element of a data frame by one or more categorical elements.

Usage

```
hierobrk(formula,data,maxlevels=10,mct=mean,lmd=NULL,umd=lmd)
```

Arguments

`formula`	A formula with a numeric element of a data frame on the left and one or more categorical elements on the right.
`data`	A data frame containing the elements in '`formula`'.
`maxlevels`	The maximum number of levels in any categorical element. Mainly to prevent the mess caused by breaking down by a huge number of categories.
`mct`	The measure of central tendency function to use - defaults to the normal standard error.
`lmd`	The lower measure of dispersion function to use.
`umd`	The upper measure of dispersion function to use.

Details

'`hierobrk`' performs the breakdown of a numeric element of a data frame by one or more categorical elements. For each category and optionally subcategories, the variable on the left of the formula is summarized as specified by the functions named in '`num.desc`'.

The user should take care when specifying different summary functions. '`hierobarp`' expects a measure of central tendency as the first function and measures of dispersion as the second and third, if "error bars" are to be displayed.

Value

A list with four elements:

`mctlist`	The array produced by the function passed as the '`mct`' argument.
`lcllist`	The array produced by the function passed as the '`lmd`' argument.
`ucllist`	The array produced by the function passed as the '`umd`' argument.
`barlabels`	A list containing the unique elements of the variables on the right side of the formula (or the levels if they are factors), in the order in which they appear in the formula. These will be the default labels for the '`hierobarp`' function.

This function is similar to '`brkdn`' in the **prettyR** package, but is structured to be used with the '`hierobarp`' function.

Author(s)

Jim Lemon

See Also

'by'

Examples

```
test.df<-data.frame(Age=rnorm(100,25,10),
 Sex=sample(c("M","F"),100,TRUE),
 Marital=sample(c("M","X","S","W"),100,TRUE),
 Employ=sample(c("FT","PT","NO"),100,TRUE))
hierobrk(formula=Age~Sex+Marital+Employ,data=test.df)
```

`intersectDiagram` *Display set intersections*

Description

Display set intersections as rows of rectangles.

Usage

```
intersectDiagram(x,pct=FALSE,show.nulls=FALSE,xnames=NULL,namesep="-",
 mar=c(0,0,3,0),main="Intersection Diagram",col=NULL,minspacing=0.1)
```

Arguments

`x`	A list containing as many numeric vectors as there are sets. The first vector contains the counts or percentages of the elements that are only in one set, the next vector contains the counts or percentages of elements that are in two sets and so on. A matrix of set membership indicators can be passed - see Details.
`pct`	Whether to display counts (FALSE) or percentages (TRUE) of the number of entities.
`show.nulls`	Whether to display the number of original objects that were not members of any set.
`xnames`	Optional user supplied names for the set categories (see Details).
`namesep`	The separator to use between category names.
`mar`	The margins for the diagram. The margins that were in effect when the function is called are restored.
`main`	The title for the diagram.
`col`	Colors for the sets (see Details).
`minspacing`	The minimum spacing between the rectangles as a proportion.

Details

'`intersectDiagram`' displays rows of optionally colored rectangles. The topmost row represents the elements that only belong to one set, the next row down represents elements belonging to two sets, and so on to the bottom row of one rectangle representing the elements that belong to all the sets. More than three intersecting sets generally produce a complex and difficult to interpret Venn diagram, and this provides an alternative way to display the intersections between larger numbers of sets.

Each set is assigned a color if '`col`' is not NA. '`rainbow`' is called if '`col`' is NULL, otherwise the colors passed are used. For each intersection, the colors representing the sets intersecting are included in the rectangle.

The strings displayed on each rectangle are taken from the argument '`xnames`' unless that is NULL, then the '`names`' of the first element of the intersectList object passed as '`x`' or returned from the call to '`makeIntersectList`' and if this is also NULL, capital letters are assigned to each category in '`x`'.

If there were objects in the original data set that were not members of any set, any percentages calculated will reflect this. By setting '`show.nulls`' to TRUE, the counts or percentages of such objects will be displayed below the intersections over an empty rectangle scaled to the count or percentage.

If a matrix of set membership indicators is passed as '`x`', it will be passed to '`makeIntersectList`' for conversion.

Value

nil

Author(s)

Jim Lemon

See Also

'`makeIntersectList`', '`getIntersectList`'

Examples

```
# create a matrix where each row represents an element and
# a 1 (or TRUE) in each column indicates that the element is a member
# of that set.
druguse<-matrix(c(sample(c(0,1),200,TRUE),
 sample(c(0,1),200,TRUE),
 sample(c(0,1),200,TRUE),
 sample(c(0,1),200,TRUE)),ncol=4)
colnames(druguse)<-c("Alc","Tob","THC","Amp")
druglist<-makeIntersectList(druguse)
# first display it as counts
intersectDiagram(druglist)
# then as percent with non.members, passing the initial matrix
intersectDiagram(druguse,pct=TRUE,show.nulls=TRUE)
```

`lengthKey` *Key for interpreting lengths in a plot*

Description

Key for interpreting lengths in a plot

Usage

```
lengthKey(x,y,tickpos,scale)
```

Arguments

`x,y`	The position of the left end of the key in user units.
`tickpos`	The labels that will appear above the key.
`scale`	A value that will scale the length of the key.

Details

'`lengthKey`' displays a line with tick marks and the values in '`tickpos`' above those tickmarks. It is useful when line segments on a plot represent numeric values. Note that if the plot does not have a 1:1 aspect ratio, a length key is usually misleading.

Value

nil

Author(s)

Jim Lemon

See Also

'`segments`', '`arrows`'

Examples

```
# manufacture a matrix of orientations in radians
o<-matrix(rep(pi*seq(0.1,0.8,by=0.1),7),ncol=8,byrow=TRUE)
m<-matrix(rnorm(56)+4,ncol=8,byrow=TRUE)
# get an empty plot of approximately 1:1 aspect ratio
plot(0,xlim=c(0.7,8.3),ylim=c(0.7,7.3),type="n")
vectorField(o,m,vecspec="rad")
# the scaling usually has to be worked out by trial and error
lengthKey(0.3,-0.5,c(0,5,10),0.24)
```

`listDepth` *Find the maximum depth of a list.*

Description

Descend a list and find the maximum number of levels.

Usage

```
listDepth(x)
```

Arguments

`x`	A list.

Details

A possibly nested list of lists is descended to determine the maximum number of levels. Currently used to set up the dimensions of a dendrite plot.

Value

The maximum number of levels in the list.

Author(s)

Jim Lemon

See Also

‘`plot.dendrite`’

`makeDendrite` *Build a list of the mutually exclusive attributes of objects.*

Description

Build a list of mutually exclusive attributes from a matrix of category indicators.

Usage

```
makeDendrite(x)
```

Arguments

`x`	A data frame or matrix where rows represent objects and columns mutually exclusive attributes of a given class.

Details

The values in ‘x’ indicate which attribute of a particular class is possessed by the object. For instance, the attributes dead and alive are mutually exclusive. ‘makeDendrite’ creates a nested list that contains the counts of successive combinations of the attributes. The top level attributes are taken from the first column, then those are combined with the attributes in the second column and so on.

Value

A list of the counts of objects for each combination of the attribute classes.

Author(s)

Jim Lemon

See Also

‘plot.dendrite’

Examples

```
sex<-sample(c("M","F"),100,TRUE)
hair<-sample(c("Blond","Black","Brown","Red"),100,TRUE)
eye<-sample(c("Blue","Black","Brown","Green"),100,TRUE)
charac<-data.frame(sex=sex,hair=hair,eye=eye)
characlist<-makeDendrite(charac)
characlist
```

---

makeIntersectList *Count set intersections*

Description

Create a list of set intersections from a matrix of indicators.

Usage

```
makeIntersectList(x,xnames=NULL)
```

Arguments

x	A data frame or matrix where rows represent objects and columns attributes. A ‘1’ or ‘TRUE’ indicates that the object (row) has that attribute or is a member of that set (column).
xnames	Optional user-supplied names for the categories of x.

Details

'`makeIntersectList`' reads a matrix (or data frame) containing only dichotomous values (either 0/1 or FALSE/TRUE). Each row represents an object and each column represents a set. A value of 1 or TRUE indicates that that object is a member of that set. The function creates a list of vectors that correspond to all combinations of the sets (set intersections) and inserts the counts of elements in each combination. If a row of the matrix is all zeros, it will not be counted, but the last element of the list returned contains the count of rows in '`x`' and thus non-members can be calculated.

makeIntersectList combines the set or attribute names to form intersection names. For the intersection of sets A and B, the name will be AB and so on. These are the names that will be displayed by intersectDiagram. To change these, use the '`xnames`' argument.

Value

A list of the intersection counts or percentages and the total number of objects.

Author(s)

Jim Lemon

See Also

'`intersectDiagram`', '`pasteCols`'

Examples

```
# create a matrix where each row represents an element and
# a 1 (or TRUE) in each column indicates that the element is a member
# of that set.
setdf<-data.frame(A=sample(c(0,1),100,TRUE,prob=c(0.7,0.3)),
 B=sample(c(0,1),100,TRUE,prob=c(0.7,0.3)),
 C=sample(c(0,1),100,TRUE,prob=c(0.7,0.3)),
 D=sample(c(0,1),100,TRUE,prob=c(0.7,0.3)))
setdflist<-makeIntersectList(setdf)
setdflist
```

`multhist` *Plot a multiple histogram, as a barplot*

Description

Given a list, plots a side-by-side barplot containing the histograms of the elements

Usage

```
multhist(x,beside=TRUE,freq=NULL,probability=!freq,plot.it=TRUE,...)
```

Arguments

`x`	a list of numeric vectors
`beside`	plot histogram bars for groups side-by-side?
`freq`	logical; if 'TRUE', the histogram graphic is a representation of frequencies, the 'counts' component of the result; if 'FALSE', probability densities, component 'density', are plotted (so that the histogram has a total area of one). Defaults to 'TRUE' if 'probability' is not specified (does not consider equidistant breaks as in '`hist`')
`probability`	an alias for '!freq', for S compatibility
`plot.it`	Whether or not to display the histogram.
`...`	additional arguments to '`hist`' or '`barplot`'

Value

the breaks and the values for the histograms.

Note

The 'inside' argument to '`barplot`' (which is not currently implemented in barplot anyway) is deleted from the argument list. The default value of NULL for '`freq`' is for consistency with '`hist`' but is equivalent to TRUE.

Author(s)

Ben Bolker

See Also

'`hist`','`barplot`'

Examples

```
l <- list(runif(10)*10,1:10,c(1,1,1,1,4,8))
multhist(l)
```

`multsymbolbox` *Draw boxes filled with symbols*

Description

Draw boxes on the current figure filled with symbols representing individual counts.

Usage

```
multsymbolbox(x1,y1,x2,y2,tot,relw=0.8,fg=par("fg"),bg=par("bg"),
 box=TRUE,debug=FALSE,...)
```

Arguments

`x1`	numeric vector: left sides of boxes
`y1`	numeric vector: bottom sides of boxes
`x2`	numeric vector: right sides of boxes
`y2`	numeric vector: top sides of boxes
`tot`	numeric vector: total numbers of symbols to put in each box
`relw`	relative width (relative to height) of symbols
`fg`	foreground color(s)
`bg`	background color(s)
`box`	(logical) draw box borders?
`debug`	debug output?
`...`	additional arguments to polygon() for drawing boxes

Value

none

Author(s)

Ben Bolker

Examples

```
plot(1:10,1:10,type="n")
multsymbolbox(c(2,4),5,c(4,5),8,tot=c(10,8))
```

`oz.windrose` *Display an Australian wind rose.*

Description

Displays a wind rose in the style used by the Australian Bureau of Meteorology.

Usage

```
oz.windrose(windagg,speed.col=c("#dab286","#fe9a66","#ce6733","#986434"),
 speed.width=c(0.2,0.4,0.6,0.8),show.legend=TRUE,legend.pos=27,...)
```

Arguments

`windagg`	A matrix of percentages with the rows representing speed ranges and the columns indicating wind directions.
`speed.col`	Colors representing speed ranges.
`speed.width`	Half widths of the bars representing speed ranges.
`show.legend`	Logical indicating whether to display a legend.
`legend.pos`	The vertical position of the wind rose legend. The Australian Bureau of Meteorology displays the legend at the top of the plot
`...`	additional arguments passed to '`plot`'.

Details

'`oz.windrose`' displays a wind rose in the style used by the Australian Bureau of Meteorology. Each limb represents a bin of wind directions, and there are conventionally eight bins. If '`windagg`' has more than eight columns, more limbs will be displayed. The rows of '`windagg`' represent the speed ranges used by the Australian Bureau of Meteorology (0, 0-10, 10-20, 20-30 and over 30 in km/hour). The diameter of the central circle is calculated as (percent calm observations)/(number of direction bins). The remaining grid circles are spaced from the circumference of the "Calm" circle.

Value

nil

Note

If a title is desired, remember to move the legend to the bottom of the plot. If the function is passed values that do not sum to 100, the resulting plot will at best be misleading.

Author(s)

Jim Lemon (thanks to Anna in the Sydney BoM office)

See Also

'`oz.windrose.legend`', '`draw.circle`', '`bin.wind.records`'

Examples

```
windagg<-matrix(c(8,0,0,0,0,0,0,0,4,6,2,1,6,3,0,4,2,8,5,3,5,2,1,1,
 5,5,2,4,1,4,1,2,1,2,4,0,3,1,3,1),nrow=5,byrow=TRUE)
oz.windrose(windagg)
```

`oz.windrose.legend` *Display an Australian wind rose legend.*

Description

Displays a wind rose legend in the style used by the Australian Bureau of Meteorology.

Usage

```
oz.windrose.legend(speed.col=c("#dab286","#fe9a66","#ce6733","#986434"),
 speed.width=c(0.2,0.4,0.6,0.8),legend.pos=27)
```

Arguments

`speed.col`	Colors representing speed ranges.
`speed.width`	Half widths of the bars representing speed ranges.
`legend.pos`	The vertical position of the wind rose legend. The Australian Bureau of Meteorology displays the legend at the top of the plot

Value

nil

Author(s)

Jim Lemon (thanks to Anna in the Sydney BoM office)

See Also

'`oz.windrose`'

Examples

```
plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",axes=FALSE,xlab="",ylab="")
par(xpd=TRUE)
oz.windrose.legend()
par(xpd=FALSE)
```

`p.ions` *Calculate ion/total(ion) ratios.*

Description

Calculates the ion/total(ions) ratios of ions used in the Piper diagram.

Usage

```
p.ions(meq,short.output=TRUE)
```

Arguments

`meq`	A data frame containing the concentrations of ions used in the Piper diagram in milliequivalents/liter.
`short.output`	Whether to return the proportions of four ions (TRUE) or include carbonate-bicarbonate and sodium-potassium (FALSE).

Details

'`p.ions`' calculates the proportions of anions or cations to total anions or cations for the ions used in the Piper diagram.

Value

The proportion of the ions.

Author(s)

Mike Cheetham

See Also

'`piper.diagram`'

`pasteCols` *Paste the columns of a matrix together.*

Description

Paste the columns of a matrix together to form as many "words" as there are columns.

Usage

```
pasteCols(x,sep="")
```

Arguments

`x`	A matrix.
`sep`	The separator to use in the ‘`paste`’ command.

Details

‘`pasteCols`’ pastes the columns of a matrix together to form a vector in which each element is the concatenation of the elements in each of the columns of the matrix. It is intended for producing identifiers from a matrix returned by the ‘`combn`’ function.

Value

A vector of character strings.

Author(s)

Jim Lemon

See Also

‘`makeIntersectList`’

Examples

```
# create a matrix of the combinations of the first five letters of the
# alphabet taken two at a time.
alpha5<-combn(LETTERS[1:5],2,simplify=TRUE)
pasteCols(alpha5,sep="+")
```

`pie.labels` *Place labels on a pie chart*

Description

Places labels on a pie chart

Usage

```
pie.labels(x,y,angles,labels,radius=1,bg="white",border=TRUE,...)
```

Arguments

`x,y`	x and y position of the center of the pie chart
`angles`	A numeric vector representing angles in radians. This is the return value of '`floating.pie`'.
`labels`	Text strings to label each sector.
`radius`	The radius at which to place the labels in user units. The default is 1.
`bg`	The color of the rectangles on which the labels are displayed.
`border`	Whether to draw borders around the rectangles.
`...`	Arguments passed to '`boxed.labels`'.

Value

nil

Note

Remember that '`x`' and '`y`' specify the center of the pie chart and that the label positions are specified by angles and radii from that center.

Author(s)

Jim Lemon

See Also

'`floating.pie`', '`boxed.labels`'

Examples

```
pieval<-c(2,4,6,8)
plot(1:5,type="n",axes=FALSE)
box()
bisect.angles<-floating.pie(3,3,pieval)
pie.labels(3,3,bisect.angles,c("two","four","six","eight"))
```

`pie3D` *Display a 3D pie chart*

Description

Displays a 3D pie chart with optional labels.

Usage

```
pie3D(x,edges=100,radius=1,height=0.3,theta=pi/6,start=0,border=par("fg"),
 col=NULL,labels=NULL,labelpos=NULL,labelcol=par("fg"),labelcex=1.5,
 sector.order=NULL,explode=0,shade=0.8,...)
```

Arguments

`x`	a numeric vector for which each value will be a sector
`edges`	the number of lines forming an ellipse
`radius`	the radius of the pie in user units
`height`	the height of the pie in user units
`theta`	The angle of viewing in radians
`start`	The angle at which to start drawing sectors.
`border`	The color of the sector border lines
`col`	The colors of the sectors
`labels`	Optional labels for each sector
`labelpos`	Optional positions for the labels
`labelcol`	The color of the labels
`labelcex`	The character expansion factor for the labels
`sector.order`	Allows the operator to specify the order in which the sectors are drawn.
`explode`	The amount to "explode" the pie in user units
`shade`	If > 0 and < 1, the proportion to reduce the brightness of the sector color to get a better 3D effect.
`...`	graphical parameters passed to '`plot`'

Details

'`pie3D`' scales the values in '`x`' so that they total 2*pi, dropping zeros and NAs. It then displays an empty plot, calculates the sequence for drawing the sectors and calls '`draw.tilted.sector`' to draw each sector. If labels are supplied, it will call '`pie3D.label`' to place a label for each sector. If supplied, the number of labels, label positions and sector colors must be at least equal to the number of values in '`x`'. If the labels are long, it may help to reduce the radius of the pie as in the example below.

Value

The bisecting angle of the sectors in radians.

Note

Due to the somewhat primitive method used to draw sectors, a sector that extends beyond both pi/2 and 3*pi/2 radians in either direction may not display properly. Setting '`start`' to pi/2 will often fix this, but the user may have to adjust '`start`' and the order of sectors in extreme cases. The argument '`sector.order`' allows the user to specify a vector of integers that will override the calculation of the order in which the sectors are drawn. This is usually necessary when a very large sector that extends past 3*pi/2 is overlapped by a smaller sector next to it.

While '`pie3D`' can be used to display a 2D pie chart by setting height=0 and theta=pi, the labels produced by '`pie3D.labels`' will not be well positioned. It is probably better to use '`floating.pie`' for this, or use '`pie.labels`' for the labels.

Author(s)

Jim Lemon

See Also

'`pie3D.labels`', '`draw.tilted.sector`'

Examples

```
pieval<-c(2,4,6,8)
pielabels<-
 c("We hate\n pies","We oppose\n  pies","We don't\n  care","We just love pies")
pie3D(pieval,radius=0.9,labels=pielabels,explode=0.1,main="3D PIE OPINIONS")
```

`pie3D.labels` *Display labels on a 3D pie chart*

Description

Displays labels on a 3D pie chart.

Usage

```
pie3D.labels(radialpos,radius=1,height=0.3,theta=pi/6,
 labels,labelcol=par("fg"),labelcex=1.5,minsep=0.3)
```

Arguments

`radialpos`	Position of the label in radians
`radius`	the radius of the pie in user units
`height`	the height of the pie in user units
`theta`	The angle of viewing in radians
`labels`	The label to display
`labelcol`	The color of the labels
`labelcex`	The character expansion factor for the labels
`minsep`	The minimum angular separation between label positions.

Details

'`pie3D.label`' displays labels on a 3D pie chart. The positions of the labels are given as angles in radians (usually the bisector of the pie sectors). As the labels can be passed directly to '`pie3D`', this function would probably not be called by the user.

'`pie3D.labels`' tries to separate labels that are placed closer than '`minsep`' radians. This simple system will handle minor crowding of labels. If labels are very crowded, capturing the return value of '`pie3D`' and editing the label positions may allow the user to avoid manually placing labels.

Value

nil

Author(s)

Jim Lemon

See Also

'`pie3D`', '`draw.tilted.sector`'

Examples

```
pieval<-c(2,4,6,8)
bisectors<-pie3D(pieval,explode=0.1,main="3D PIE OPINIONS")
pielabels<-
 c("We hate\n pies","We oppose\n  pies","We don't\n  care","We just love pies")
pie3D.labels(bisectors,labels=pielabels)
```

`piper.coords` *Calculate point coordinates for a Piper diagram.*

Description

Calculate point coordinates for a Piper diagram.

Usage

```
piper.coords(ions,plot.sep=0.15)
```

Arguments

`ions`	data frame containing ion concentrations.
`plot.sep`	The offsets for the ions.

Value

A data frame containing the coordinates of the points.

Author(s)

Mike Cheetham

See Also

'`piper.diagram`'

piper.diagram *Display a Piper diagram.*

Description

Displays a Piper diagram.

Usage

```
piper.diagram(ca,mg,so4,cl,ions=data.frame(ca=ca,mg=mg,so4=so4,cl=cl),
  sites=1:NROW(ions),new=FALSE,ppm=TRUE,chull=FALSE,tcsep=0.2,pch=3,
  main="",cex.lab=0.7,cex.tck=0.6,cex.pch=1,grid=TRUE,col=NA,ticklength=0.03,
  lwd.frame=1,pch.lwd=0.7,lwd.grid=lwd.frame,col.box="black",col.tck=col.box,
  col.grid="grey")
```

Arguments

`ca,mg,so4,cl`	Concentrations of the ions.
`ions`	data frame containing the above values.
`sites`	Indices for the ions.
`new`	Whether this is a new diagram.
`ppm`	Whether the concentrations are in milligrams/liter (parts per million) or milliequivalents.
`chull`	Whether to use the ‘`chull`’ function.
`tcsep`	Spacing for tick mark labels.
`pch`	Symbol to use for ion values.
`main`	Title for the diagram.
`cex.lab`	Expansion for the label text.
`cex.tck`	Expansion for the tick text.
`cex.pch`	Expansion for the point text.
`grid`	Whether to display a grid.
`col`	Colors for the points.
`ticklength`	Length for the ticks.
`lwd.frame`	Line width for the frame.
`pch.lwd`	Line width for the symbols.
`lwd.grid`	Line width for the grid.
`col.box`	Color for the box.
`col.tck`	Color for the ticks.
`col.grid`	Color for the grid.

Value

The return value from piper.points.

Author(s)

Mike Cheetham

`piper.points` *Display the points on a Piper diagram.*

Description

Displays the points for the anions and cations in the Piper diagram.

Usage

```
piper.points(ca,mg,so4,cl,ions=data.frame(ca=ca,mg=mg,so4=so4,cl=cl),sites,
  ppm=TRUE,chull=FALSE,pch=3,main="",cex.pch=1,col=NA,pch.lwd=0.7,plot.sep=0.15)
```

Arguments

`ca,mg,so4,cl`	Concentrations of the ions.
`ions`	data frame containing the above values.
`sites`	indices for ions.
`ppm`	Whether the concentrations are in milligrams/liter (parts per million) or milliequivalents.
`chull`	Whether to display points or polygons for the concentrations.
`pch`	Symbol to use in plotting points.
`main`	title for the diagram
`cex.pch`	Expansion for the points.
`col`	Colors for the points/polygons.
`pch.lwd`	Line widths.
`plot.sep`	The offsets for the ions.

Details

'`piper.points`' displays the points/polygons in the Piper diagram.

Value

A list containing the coordinates of the points and optionally information about the indices and colors.

Author(s)

Mike Cheetham

See Also

'`piper.diagram`'

`piper.set` *Calculate ion offsets.*

Description

Calculates the offsets for anions and cations in the Piper diagram.

Usage

```
piper.set(plot.sep=0.15)
```

Arguments

`plot.sep`	The offsets of the ions in user units.

Details

'`piper.set`' calculates the ion offsets in the Piper diagram.

Value

The offsets.

Author(s)

Mike Cheetham

See Also

'`piper.diagram`'

`piper.text` *Display a string on a Piper diagram.*

Description

Displays a character string on the Piper diagram.

Usage

```
piper.text(ca,mg,so4,cl,ions=data.frame(ca=ca,mg=mg,so4=so4,cl=cl),ptext,
  ppm=TRUE,cex=1,col=1,plot.sep=0.15,pos=4)
```

Arguments

`ca,mg,so4,cl`	Concentrations of the ions.
`ions`	data frame containing the above values.
`ptext`	The character string.
`ppm`	Whether the concentrations are in milligrams/liter (parts per million) or milliequivalents.
`cex`	Expansion for the text.
`col`	Colors for the text.
`plot.sep`	The offsets.
`pos`	The position of the string relative to the data point.

Value

nil

Author(s)

Mike Cheetham

See Also

‘`piper.diagram`’

plot.dendrite *Plot a dendrogram of a dendrite object.*

Description

Plot a dendrogram for two or more mutually exclusive attributes.

Usage

```
## S3 method for class 'dendrite':
plot(x,xlabels=NULL,main="",mar=c(1,0,3,0),cex=1,...)
```

Arguments

`x`	A '`dendrite`' object containing the counts of objects having combinations of mutually exclusive attributes.
`xlabels`	The category labels that will be displayed beneath the dendrogram.
`main`	The title of the plot.
`mar`	Margins for the plot.
`cex`	Character expansion for the leaves of the dendrogram.
`...`	Additional arguments passed to '`plot`'.

Details

'`plot.dendrite`' sets up a plot for a dendrogram. The actual plotting of the dendrogram is done by '`furc`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`furc`'

Examples

```
sex<-sample(c("M","F"),100,TRUE)
hair<-sample(c("Blond","Black","Brown","Red"),100,TRUE)
eye<-sample(c("Blue","Black","Brown","Green"),100,TRUE)
charac<-data.frame(sex=sex,hair=hair,eye=eye)
characlist<-makeDendrite(charac)
plot.dendrite(characlist,names(charac),main="Test dendrogram",cex=0.8)
```

`plotCI` *Plot confidence intervals/error bars*

Description

Given a set of x and y values and upper and lower bounds, this function plots the points with error bars.

Usage

```
plotCI(x,y=NULL,uiw,liw=uiw,ui=NULL,li=NULL,err="y",
  sfrac=0.01,gap=0,slty=par("lty"),add=FALSE,scol=NULL,pt.bg=par("bg"),...)
```

Arguments

`x`	The x coordinates of points in the plot
`y`	The y coordinates of points in the plot
`uiw`	The width of the upper portion of the confidence region, or (if '`liw`' is missing) the width of both halves of the confidence region
`liw`	The width of the lower portion of the confidence region (if missing, the function assumes symmetric confidence bounds)
`ui`	The absolute upper limit of the confidence region
`li`	The absolute lower limit of the confidence region
`err`	The direction of error bars: "x" for horizontal, "y" for vertical ("xy" would be nice but is not implemented yet; don't know quite how everything would be specified. See examples for composing a plot with simultaneous horizontal and vertical error bars)
`gap`	Size of gap in error bars around points (default 0;gap=TRUE gives gap size of 0.01)
`sfrac`	Scaling factor for the size of the "serifs" (end bars) on the confidence bars, in x-axis units
`add`	If FALSE (default), create a new plot; if TRUE, add error bars to an existing plot.
`slty`	Line type of error bars
`scol`	Color of error bars: if '`col`' is specified in the optional arguments, '`scol`' is set the same; otherwise it's set to '`par(scol)`'
`pt.bg`	Background color of points (use pch=21, pt.bg=par("bg") to get open points superimposed on error bars)
`...`	Any other parameters to be passed through to '`plot.default`', '`points`', '`arrows`', etc. (e.g. '`lwd`', '`col`', '`pch`', '`axes`', '`xlim`', '`ylim`'). '`xlim`' and '`ylim`' are set by default to include all of the data points and error bars. '`xlab`' and '`ylab`' are set to the names of '`x`' and '`y`'. If '`pch==NA`', no points are drawn (e.g. leaving room for text labels instead)

Value

invisible(x,y); creates a plot on the current device.

Author(s)

Ben Bolker (documentation and tweaking of a function provided by Bill Venables, additional feature ideas from Gregory Warnes)

See Also

'`boxplot`'

Examples

```
y<-runif(10)
err<-runif(10)
plotCI(1:10,y,err)
plotCI(1:10,y,err,2*err,lwd=2,col="red",scol="blue")
err.x<-runif(10)
err.y<-runif(10)
plotCI(1:10,y,err.y,pt.bg=par("bg"),pch=21)
plotCI(1:10,y,err.x,pt.bg=par("bg"),pch=21,err="x",add=TRUE)
data(warpbreaks)
attach(warpbreaks)
wmeans<-by(breaks,tension,mean)
wsd<-by(breaks,tension,sd)
## note that barplot() returns the midpoints of the bars, which plotCI
##  uses as x-coordinates
plotCI(barplot(wmeans,col="gray",ylim=c(0,max(wmeans+wsd))),wmeans,wsd,add=TRUE)
## using labels instead of points
labs<-sample(LETTERS,replace=TRUE,size=10)
plotCI(1:10,y,err,pch=NA,gap=0.02)
text(1:10,y,labs)
```

`polar.plot` *Plot values on a circular grid of 0 to 360 degrees.*

Description

'`polar.plot`' displays a plot of radial lines, symbols or a polygon centered at the midpoint of the plot frame on a 0:360 circle. Positions are interpreted as beginning at the right and moving counterclockwise unless '`start`' specifies another starting point or '`clockwise`' is TRUE.

Usage

```
polar.plot(lengths,polar.pos=NULL,labels,label.pos=NULL,
 start=0,clockwise=FALSE,rp.type="r",...)
```

Arguments

`lengths`	numeric data vector. Magnitudes will be represented as the radial positions of symbols, line ends or polygon vertices.
`polar.pos`	numeric vector of positions on a 0:360 degree circle. These will be converted to radians when passed to '`radial.plot`'.
`labels`	text labels to place on the periphery of the circle. This defaults to labels every 20 degrees. For no labels, pass an empty string.
`label.pos`	positions of the peripheral labels in degrees
`start`	The position for zero degrees on the plot in degrees.
`clockwise`	Whether to increase angles clockwise rather than the default counterclockwise.
`rp.type`	Whether to plot radial lines, symbols or a polygon.
`...`	additional arguments passed to '`radial.plot`' and then to '`plot`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`radial.plot`'

Examples

```
testlen<-c(rnorm(36)*2+5)
testpos<-seq(0,350,by=10)
polar.plot(testlen,testpos,main="Test Polar Plot",lwd=3,line.col=4)
polar.plot(testlen,testpos,main="Test Clockwise Polar Plot",
 start=90,clockwise=TRUE,lwd=3,line.col=4)
# reset the margins
par(mar=c(5,4,4,2))
```

`polygon.shadow` *Display a shadow effect for an arbitrary polygon*

Description

Displays a shadow effect on an existing plot

Usage

```
polygon.shadow(x,y=NULL,offset=NA,inflate=NA,col=c("#ffffff","#cccccc"))
```

Arguments

`x,y`	x and y coordinate of the vertices of the polygon. ‘`y`’ can be missing if ‘`x`’ is a list with ‘`x`’ and ‘`y`’ components.
`offset`	a vector containing the values of the x and y offsets for the shadow. Defaults to 1/20 of the maximum x and y dimensions of the polygon.
`col`	the colors of the shadow from the outer edge to the central part.
`inflate`	the amount to "inflate" the shadow relative to the polygon (i.e. the penumbra). Defaults to the values in ‘`offset`’.

Details

‘`polygon.shadow`’ is typically called just before drawing a polygon. It displays a shadow effect by drawing the polygon ten times, beginning with the first color in ‘`col`’ and stepping through to the second color to create a "shadow" (or a "halo" if you prefer). Each successive polygon is shrunk by 10% of ‘`inflate`’. The default shadow effect has the light at the upper left. This effect may also be used as a text background.

Value

nil

Note

The background must be a constant color or the shadow effect will not look right.

Author(s)

Jim Lemon

See Also

‘`polygon`’

Examples

```
par(pty="s")
plot(1:5,type="n",main="Polygon Shadow test",xlab="",ylab="",axes=FALSE)
box()
# do a shadow on a yellow square
polygon(c(1,2.2,2.2,1),c(5,5,3.8,3.8),col="#ffff00")
polygon.shadow(c(1.2,2,2,1.2),c(4.8,4.8,4,4),col=c("#ffff00","#cccc00"))
polygon(c(1.2,2,2,1.2),c(4.8,4.8,4,4),col=c("#ff0000"))
# a green triangle on a light blue square with a big offset
polygon(c(4,5,5,4),c(2,2,1,1),col="#aaaaff")
polygon.shadow(c(4.5,4.8,4.2),c(1.7,1.2,1.2),col=c("#aaaaff","#8888cc"),
 offset=c(0.1,-0.1),inflate=c(0.2,0.2))
polygon(c(4.5,4.8,4.2),c(1.7,1.2,1.2),col=c("#00ff00"))
# now a circle as a background
polygon.shadow(cos(seq(0,2*pi,by=pi/20))+3,sin(seq(0,2*pi,by=pi/20))+3,
 offset=c(0,0),inflate=c(0.1,0.1))
```

```
text(3,3,"Polygon shadow\nas a circular\ntext background",cex=1.5)
```

`pyramid.plot` *Pyramid plot*

Description

Displays a pyramid (opposed horizontal bar) plot on the current graphics device.

Usage

```
pyramid.plot(xy,xx,labels,top.labels=c("Male","Age","Female"),
 main="",laxlab=NULL,raxlab=NULL,unit="%",xycol,xxcol,gap=1,
 labelcex=1,mark.cat=NA,add=FALSE)
```

Arguments

`xy,xx`	Vectors of percentages (but see Details) both of which should total 100 if this is a population pyramid, and be of equal length.
`labels`	Labels for the categories represented by each pair of bars. There should be a label for each xy or xx value, even if empty.
`top.labels`	The two categories represented on the left and right sides of the plot and a heading for the labels up the center.
`main`	Optional title for the plot.
`laxlab`	Optional labels for the left x axis ticks.
`raxlab`	Optional labels for the right x axis ticks.
`unit`	The label for the units of the plot.
`xycol,xxcol`	Color(s) for the left and right sets of bars. Both of these default to 1:length(labels).
`gap`	One half of the space between the two sets of bars for the '`labels`' in user units.
`labelcex`	Expansion for the category labels.
`mark.cat`	If an integer equal to the index of one of the labels is passed, that label will have a rectangle drawn around it.
`add`	Whether to add bars to an existing plot. Usually this involves overplotting a second set of bars, perhaps transparent.

Details

'`pyramid.plot`' is principally intended for population pyramids, although it can display other types of opposed bar charts with suitable modification of the arguments. If the user wants a different unit for the display, just change '`unit`' accordingly. The default gap of two units is usually satisfactory for the four to six percent range of most bars on population pyramids.

If '`xy`' and '`xx`' are matrices or data frames, '`pyramid.plot`' will produce opposed stacked bars with the first columns innermost. In this mode, colors are limited to one per column. The stacked bar mode will in general not work with the '`add`' method. Note that the stacked bar mode can get very messy very quickly.

The '`add`' argument allows one or more sets of bars to be plotted on an existing plot. If these are not transparent, any bar that is shorter than the bar that overplots it will disappear. Only some graphic devices (e.g. '`pdf`') will handle transparency.

In order to add bars, the function cannot restore the initial margin values or the new bars will not plot properly. To automatically restore the plot margins, call the function as in the example.

Value

The return value of '`par("mar")`' when the function was called.

Author(s)

Jim Lemon

See Also

'`rect`'

Examples

```
xy.pop<-c(3.2,3.5,3.6,3.6,3.5,3.5,3.9,3.7,3.9,3.5,3.2,2.8,2.2,1.8,
 1.5,1.3,0.7,0.4)
xx.pop<-c(3.2,3.4,3.5,3.5,3.5,3.7,4,3.8,3.9,3.6,3.2,2.5,2,1.7,1.5,
 1.3,1,0.8)
agelabels<-c("0-4","5-9","10-14","15-19","20-24","25-29","30-34",
 "35-39","40-44","45-49","50-54","55-59","60-64","65-69","70-74",
 "75-79","80-44","85+")
xycol<-color.gradient(c(0,0,0.5,1),c(0,0,0.5,1),c(1,1,0.5,1),18)
xxcol<-color.gradient(c(1,1,0.5,1),c(0.5,0.5,0.5,1),c(0.5,0.5,0.5,1),18)
par(mar=pyramid.plot(xy.pop,xx.pop,labels=agelabels,
 main="Australian population pyramid 2002",xycol=xycol,xxcol=xxcol))
# three column matrices
avtemp<-c(seq(11,2,by=-1),rep(2:6,each=2),seq(11,2,by=-1))
malecook<-matrix(avtemp+sample(-2:2,30,TRUE),ncol=3)
femalecook<-matrix(avtemp+sample(-2:2,30,TRUE),ncol=3)
# use a background color
par(bg="#eedd55")
# group by age
agegrps<-c("0-10","11-20","21-30","31-40","41-50","51-60",
 "61-70","71-80","81-90","91+")
oldmar<-pyramid.plot(malecook,femalecook,labels=agegrps,
 unit="Bowls per month",xycol=c("#ff0000","#eeee88","#0000ff"),
 xxcol=c("#ff0000","#eeee88","#0000ff"),
 top.labels=c("Males","Age","Females"),gap=3)
# put a box around it
box()
# give it a title
mtext("Porridge temperature by age and sex of cook",3,2,cex=1.5)
# stick in a legend
```

```
legend(par("usr")[1],11,c("Too hot","Just right","Too cold"),
 fill=c("#ff0000","#eeee88","#0000ff"))
# don't forget to restore the margins and background
par(mar=oldmar,bg="transparent")
```

`qt.plot` *Quantity by interval plot*

Description

Display the counts of numeric quantities against the intervals between quantities that are equal when rounded to integers.

Usage

```
qt.plot(qnt,qtime=NA,col=NULL,border="lightgray",
 main="Quantity x interval",xlab="Interval",ylab="Quantity",
 mar=c(5,4,4,4),...)
```

Arguments

`qnt`	Numeric vector
`qtime`	Numeric vector - may be a date as an integer.
`col`	The colors to fill the strips. NA for none.
`border`	border color for the polygons
`main`	The title of the plot.
`xlab,ylab`	Axis labels.
`mar`	margins for the plot - defaults to leave space for scale
`...`	additional arguments passed to '`plot`'.

Details

The intervals calculated from '`qtime`' are the x values and the counts of values of '`qnt`' are the y values of the plot displayed as the widths of sections of polygons running across the time intervals. This plot was devised to display the distribution of drinking, but may be useful for any situation in which it is desired to display the distribution of numerically coded quantities against the intervals between their occurrence. Note that if there are many values and many intervals, the resulting plot will be mostly empty. Categorizing the values and intervals so that there are only three or four categories will often produce a more informative plot.

'`qt.plot`' assumes that the values in '`qtime`' represent interpretable intervals like seconds or days. The default is to assume sequential time intervals. If '`qtime`' contains dates, they must be translated to numeric format. These values will be sorted by the function. If '`qtime`' is NA, it will be assigned '`1:length(qnt)`'.

Value

nil

Author(s)

Jim Lemon

See Also

'`polygon`'

Examples

```
# first a moderate drinker with frequent bigger sessions
qnt<-sample(0:5,365,TRUE,prob=c(0.02,0.1,0.4,0.3,0.1,0.08))
qtdates<-seq(as.Date("2007-01-01"),as.Date("2007-12-31"),by=1)
qt.plot(qnt,as.numeric(qtdates),xlab="Number of days interval",
 ylab="Standard drinks per session")
# now add monthly bigger sessions and notice how this
qnt[c(30,60,90,120,150,180,210,240,270,300,330,360)]<-rep(4:5,length.out=12)
qt.plot(qnt,as.numeric(qtdates),xlab="Number of days interval",
 ylab="Standard drinks per session")
```

`radial.plot` *Plot values on a circular grid of 0 to 2*pi radians.*

Description

Plot numeric values as distances from the center of a circular field in the directions defined by angles in radians.

Usage

```
radial.plot(lengths,radial.pos=NULL,labels,label.pos=NULL,radlab=FALSE,
start=0,clockwise=FALSE,rp.type="r",label.prop=1.1,main="",xlab="",ylab="",
line.col=par("fg"),lty=par("lty"),lwd=par("lwd"),mar=c(2,2,3,2),
show.grid=TRUE,show.grid.labels=TRUE,show.radial.grid=TRUE,
grid.col="gray",grid.bg="transparent",
grid.left=FALSE,grid.unit=NULL,point.symbols=NULL,point.col=NULL,
show.centroid=FALSE,radial.lim=NULL,radial.labels=NULL,poly.col=NULL,...)
```

Arguments

`lengths`	A numeric data vector or matrix. If '`lengths`' is a matrix, the rows will be considered separate data vectors.
`radial.pos`	A numeric vector or matrix of positions in radians. These are interpreted as beginning at the right (0 radians) and moving counterclockwise. If '`radial.pos`' is a matrix, the rows must correspond to rows of '`lengths`'.

`labels`	Character strings to be placed at the outer ends of the lines. If set to NA, will suppress printing of labels, but if missing, the radial positions will be used.
`label.pos`	The positions of the labels around the plot in radians.
`radlab`	Whether to rotate the outer labels to a radial orientation.
`start`	Where to place the starting (zero) point. Defaults to the 3 o'clock position.
`clockwise`	Whether to interpret positive positions as clockwise from the starting point. The default is counterclockwise.
`rp.type`	Whether to draw (r)adial lines, a (p)olygon, (s)ymbols or some combination of these. If '`lengths`' is a matrix and rp.type is a vector, each row of '`lengths`' can be displayed differently.
`label.prop`	The label position radius as a proportion of the maximum line length.
`main`	The title for the plot.
`xlab,ylab`	Normally x and y axis labels are suppressed.
`line.col`	The color of the radial lines or polygons drawn.
`lty`	The line type(s) to be used for polygons or radial lines.
`lwd`	The line width(s) to be used for polygons or radial lines.
`mar`	Margins for the plot. Allows the user to leave space for legends, long labels, etc.
`show.grid`	Logical - whether to draw a circular grid.
`show.grid.labels`	Logical - whether to display labels for the grid.
`show.radial.grid`	Whether to draw radial lines to the plot labels.
`grid.col`	Color of the circular grid.
`grid.bg`	Fill color of above.
`grid.left`	Whether to place the radial grid labels on the left side.
`grid.unit`	Optional unit description for the grid.
`point.symbols`	The symbols for plotting (as in pch).
`point.col`	Colors for the symbols.
`show.centroid`	Whether to display a centroid.
`radial.lim`	The range of the grid circle. Defaults to '`pretty(range(lengths))`', but if more than two values are passed, the exact values will be displayed.
`radial.labels`	Optional labels for the radial grid. The default is the values of radial.lim.
`poly.col`	Fill color if polygons are drawn. Use NA for no fill.
`...`	Additional arguments are passed to '`plot`'.

Details

'radial.plot' displays a plot of radial lines, polygon(s), symbols or a combination of these centered at the midpoint of the plot frame, the lengths, vertices or positions corresponding to the numeric magnitudes of the data values. If 'show.centroid' is TRUE, an enlarged point at the centroid of values is displayed. The centroid is calculated as the average of x and y values unless 'rp.type="p"'. In this case, the barycenter of the polygon is calculated. Make sure that these suit your purpose, otherwise calculate the centroid that you really want and add it with the 'points' function.

If the user wants to plot several sets of lines, points or symbols by passing matrices or data frames of 'lengths' and 'radial.pos', remember that these will be grouped by row, so transpose if the data are grouped by columns.

The size of the labels on the outside of the plot can be adjusted by setting 'par(cex.axis=)' and that of the labels inside by setting 'par(cex.lab=)'. If 'radlab' is TRUE, the labels will be rotated to a radial alignment. This may help when there are many values and labels. If some labels are still crowded, try running 'label.pos' through the 'spreadout' function.

The radial.plot family of plots is useful for illustrating cyclic data such as wind direction or speed (but see 'oz.windrose' for both), activity at different times of the day, and so on. While 'radial.plot' actually does the plotting, another function is usually called for specific types of cyclic data. Note that if the observations are not taken at equal intervals around the circle, the centroid may not mean much.

Value

nil

Author(s)

Jim Lemon - thanks to Jeremy Claisse and Antonio Hernandez Matias for the 'lty' and 'rp.type' suggestions respectively, Patrick Baker for the request that led to 'radlab' and Thomas Steiner for the request for the 'radial.lim' and 'radial.labels' modifications.

See Also

'polar.plot','clock24.plot'

Examples

```
testlen<-rnorm(10)*2+5
testpos<-seq(0,18*pi/10,length=10)
testlab<-letters[1:10]
oldpar<-radial.plot(testlen,testpos,main="Test Radial Lines",line.col="red",lwd=3)
testlen<-c(sin(seq(0,1.98*pi,length=100))+2+rnorm(100)/10)
testpos<-seq(0,1.98*pi,length=100)
radial.plot(testlen,testpos,rp.type="p",main="Test Polygon",line.col="blue")
# now do a 12 o'clock start with clockwise positive
radial.plot(testlen,testpos,start=pi/2,clockwise=TRUE,
 rp.type="s",main="Test Symbols (clockwise)",
 point.symbols=16,point.col="green",show.centroid=TRUE)
# one without the circular grid and multiple polygons
# see the "diamondplot" function for variation on this
posmat<-matrix(sample(2:9,30,TRUE),nrow=3)
radial.plot(posmat,labels=paste("X",1:10,sep=""),rp.type="p",
 main="Spiderweb plot",line.col=2:4,show.grid=FALSE,lwd=1:3,
 radial.lim=c(0,10))
# dissolved ions in water
ions<-c(3.2,5,1,3.1,2.1,4.5)
ion.names<-c("Na","Ca","Mg","Cl","HCO3","SO4")
radial.plot(ions,labels=ion.names,rp.type="p",main="Dissolved ions in water",
 grid.unit="meq/l",radial.lim=c(0,5),poly.col="yellow")
par(xpd=oldpar$xpd,mar=oldpar$mar,pty=oldpar$pty)
# reset the margins
par(mar=c(5,4,4,2))
```

`rescale` *Scale numbers into a new range.*

Description

Scale a vector or matrix of numbers into a new range.

Usage

```
rescale(x,newrange)
```

Arguments

`x`	A numeric vector, matrix or data frame.
`newrange`	The minimum and maximum value of the range into which '`x`' will be scaled.

Details

'`rescale`' performs a simple linear conversion of '`x`' into the range specified by '`newrange`'. Only numeric vectors, matrices or data frames with some variation will be accepted. NAs are now preserved - formerly the function would fail.

Value

On success, the rescaled object, otherwise the original object.

Author(s)

Jim Lemon

Examples

```
# scale one vector into the range of another
normal.counts<-rnorm(100)
normal.tab<-tabulate(cut(normal.counts,breaks=seq(-3,3,by=1)))
normal.density<-rescale(dnorm(seq(-3,3,length=100)),range(normal.tab))
# now plot them
plot(c(-2.5,-1.5,-0.5,0.5,1.5,2.5),normal.tab,xlab="X values",
 type="h",col="green")
lines(seq(-3,3,length=100),normal.density,col="blue")
```

`revaxis` *Plot with axis direction(s) reversed.*

Description

Reverses the sense of either or both the ‘`x`’ and ‘`y`’ axes.

Usage

```
revaxis(x, y, xrev=FALSE, yrev=TRUE, xside=if (yrev) 3 else 1,
        yside=if (xrev) 4 else 2, xlab=NULL, ylab=NULL, bty=NULL, ...)
```

Arguments

`x`	Vector of ‘`x`’-coordinates of the data to be plotted.
`y`	Vector of ‘`y`’-coordinates of the data to be plotted.
`xrev`	Logical scalar; should the sense of the ‘`x`’-axis be reversed?
`yrev`	Logical scalar; should the sense of the ‘`y`’-axis be reversed?
`xside`	The side of the plot on which the ‘`x`’-axis labels should go.
`yside`	The side of the plot on which the ‘`y`’-axis labels should go.
`xlab`	Character string for labelling the ‘`x`’-axis.
`ylab`	Character string for labelling the ‘`y`’-axis.
`bty`	Single letter indicating the type of box to be drawn around the plot. See ‘`par()`’ for the possible letters and their meaning.
`...`	Other arguments to be passed to plot.

Value

nil

Author(s)

Rolf Turner

See Also

'plot', 'box', 'par'

Examples

```
x <- runif(20)
y <- runif(20)
revaxis(x,y,yside=4)
```

---

sizeplot *Plot with repeated symbols by size*

Description

Plots a set of (x,y) data with repeated points denoted by larger symbol sizes

Usage

```
 sizeplot(x, y, scale=1, pow=0.5, powscale=TRUE, size=c(1,4), add=FALSE, ...)
```

Arguments

`x`	x coordinates of data
`y`	y coordinates of data
`scale`	scaling factor for size of symbols
`pow`	power exponent for size of symbols
`powscale`	(logical) use power scaling for symbol size?
`size`	(numeric vector) min and max size for scaling, if powscale=FALSE
`add`	(logical) add to an existing plot?
`...`	other arguments to '`plot()`' or '`points()`'

Details

Most useful for plotting (e.g.) discrete data, where repeats are likely. If all points are repeated equally, gives a warning. The size of a point is given by $scale * n^p ow$, where n is the number of repeats, if powscale is TRUE, or it is scaled between size[1] and size[2], if powscale is FALSE.

Value

A plot is produced on the current device, or points are added to the current plot if '`add=TRUE`'.

Author(s)

Ben Bolker

See Also

'symbols'

Examples

```
x <- c(0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.2,0.2,0.3,0.3)
y <- c( 1,  1,  1,  1,  2,  2,  2,  3,  3,  4,  5 )
plot(x,y)
sizeplot(x,y)
sizeplot(x,y,pch=2)
```

---

sizetree *Display a hierarchical breakdown of disjunct categories*

Description

Display a data frame in which the values in each successive column represent subcategories of the previous column as stacked rectangles.

Usage

```
sizetree(x,left=0,top,right=1,lastcenter=NA,showval=TRUE,showcount=TRUE,
 firstcall=TRUE,col=NA,colorindex=1,...)
```

Arguments

`x`	A data frame in which each successive column represents subcategories of the previous column.
`left`	The left edge of the current stack of rectangles in user units.
`top`	The top of the current stack of rectangles in user units.
`right`	The right edge of the current stack of rectangles in user units.
`lastcenter`	The center of the previous rectangle from which the next breakdown of categories arises. There is almost no reason to change it.
`showval`	Whether to display the values representing the categories.
`showcount`	Whether to display the count for the categories.
`firstcall`	A flag for the function - do not alter this.
`col`	Optional fill colors for the rectangles. See Details
`colorindex`	The index for '`col`' if it is a list of color vectors.
`...`	additional arguments passed to '`plot`'.

Details

'`sizetree`' displays disjunct hierarchical categories as stacked rectangles. It accepts a data frame in which the values in the first column represent categories, the values in the second column represent subcategories of the first column, and so on. The first column will be displayed as a stack of rectangles, the height of each proportional to the count for each category. Each substack of rectangles in the second stack will represent the breakdown of counts for its superordinate category and so on through the columns. Empty categories are ignored and NAs will produce gaps.

Typically, the user will simply pass the data frame, which should only contain columns that are hierarchical categories, set '`showval`' and '`showcount`' to the desired values, and pass colors for the top level categories if these are wanted. If colors are passed, it is best to have at least as many colors as there are categories in the first column or some will be recycled. If different colors are desired for different levels, as in the example, pass a list of colors.

The '`firstcall`' argument is necessary for the function to initialize the plot, as each breakdown involves a recursive call. If it is changed, the best that can be expected is an uninformative plot.

Value

nil

Author(s)

Jim Lemon

See Also

'`plot`','`smoothColors`'

Examples

```
cat1<-sample(c("None","Low","Medium","High"),40,TRUE)
cat2<-sample(c("None","Low","Medium","High"),40,TRUE)
cat3<-sample(c("None","Low","Medium","High"),40,TRUE)
hcats<-data.frame(cat1,cat2,cat3)
bhcol<-list(c("#ff8080","#dddd80","#80ff80","#8080ff"),
 c("red","green","lightblue","yellow"),
 c("#ffffff","#bbbbbb","#999999","#666666"))
sizetree(hcats,col=bhcol,main="Hierarchical count chart")
```

`smoothColors` *Build a vector of color values.*

Description

'`smoothColors`' calculates a sequence of colors. If two color names in the arguments are separated by a number, that number of interpolated colors will be inserted between the two color endpoints. Any number of color names and integers may be passed, but the last argument must be a color name. If more than one integer appears between two color names, only the first will be used in the interpolation and the others will be ignored.

Usage

```
smoothColors(...,alpha=NA)
```

Arguments

`...`	an arbitrary sequence of color names and integers beginning and ending with a color name.
`alpha`	optional '`alpha`' (transparency) value.

Value

A vector of hexadecimal color values as used by '`col`'.

Author(s)

Barry Rowlingson

See Also

'`color.gradient`','`rgb`'

Examples

```
plot(1:10,main="Test opaque colors",type="n",axes=FALSE)
box()
rect(1:7,1:7,3:9,3:9,col=smoothColors("red",2,"green",2,"blue"))
```

`soil.texture` *Soil texture triangle plot*

Description

Display a USDA soil texture triangle with optional grid, labels and soil texture points.

Usage

```
soil.texture(soiltexture=NULL, main="", at=seq(0.1, 0.9, by=0.1),
             axis.labels=c("percent sand", "percent silt",
                           "percent clay"),
             tick.labels=list(l=seq(10, 90, by=10), r=seq(10, 90, by=10),
                              b=seq(10, 90, by=10)),
             show.names=TRUE, show.lines=TRUE, col.names="gray",
             bg.names=par("bg"), show.grid=FALSE, col.axis="black",
             col.lines="gray", col.grid="gray", lty.grid=3,
             show.legend=FALSE, label.points=FALSE, point.labels=NULL,
             col.symbols="black", pch=par("pch"), ...)
```

Arguments

`soiltexture`	Matrix of soil textures where each row is a soil sample and three columns contain the proportions of the components sand, silt and clay in the range 0 to 1 or percentages in the range 0 to 100.
`main`	The title of the soil texture plot. Defaults to nothing.
`at`	Positions on the three axes where ticks will be drawn.
`axis.labels`	Labels for the axes.
`tick.labels`	The tick labels for the three axes.
`show.names`	Logical - whether to show the names of different soil types within the soil triangle.
`show.lines`	Logical - whether to show the boundaries of the different soil types within the soil triangle.
`col.names`	Color of the soil names. Defaults to gray.
`bg.names`	Color to use when drawing a blank patch for the names of soil types.
`show.grid`	Logical - whether to show grid lines at each 10 level of each soil component.
`col.axis`	Color of the triangular axes, ticks and labels.
`col.lines`	Color of the boundary lines. Defaults to gray.
`col.grid`	Color of the grid lines. Defaults to gray.
`lty.grid`	Type of line for the grid. Defaults to dashed.
`show.legend`	Logical - whether to display a legend.
`label.points`	Logical - whether to call '`thigmophobe.labels`' to label the points.
`point.labels`	Optional labels for the points or legend.
`col.symbols`	Color of the symbols representing each value.
`pch`	Symbols to use in plotting values.
`...`	Additional arguments passed to '`triax.points`' and then '`points`'.

Details

'`soil.texture`' displays a triangular plot area on which soil textures defined as proportions of sand, silt and clay can be plotted. Optional grid, vertex labels, soil type divisions and names may also be displayed. If a matrix of soil textures is present, these will be plotted.

Value

If '`soiltexture`' was included, a list of the '`x,y`' positions of the soil types plotted. If not, nil.

Note

This is now a special case of '`triax.plot`'.

Author(s)

Sander Oom, Jim Lemon, and Michael Toews

References

U.S. Department of Agriculture, Natural Resources Conservation Service, 2007. *National Soil Survey Handbook*, title 430-VI.// http://soils.usda.gov/technical/handbook/

U.S. Department of Agriculture, Natural Resources Conservation Service, 2007. *Soil Texture Calculator*// http://soils.usda.gov/technical/aids/investigations/texture/

See Also

'get.soil.texture', 'triax.plot'

Examples

```
data(soils)
soil.texture(main="NO DATA")
soil.texture(soils, main="DEFAULT", pch=2)
soil.texture(soils, main="LINES AND NAMES", show.lines=TRUE,
 show.names=TRUE, pch=3)
soiltex.return<-soil.texture(soils[1:6,], main="GRID AND LEGEND",
 show.grid=TRUE, pch=4, col.symbols=1:6, show.legend=TRUE)
par(soiltex.return$oldpar)
```

---

soil.texture.uk *Soil texture triangle plot using UK conventions*

Description

Display a UK style soil texture triangle with optional grid, labels and soil texture points.

Usage

```
soil.texture.uk(soiltexture = NULL, main = "",at = seq(0.1, 0.9, by = 0.1),
 axis.labels = c("percent sand", "percent silt", "percent clay"),
 tick.labels = list(l = seq(10, 90, by = 10), r = seq(10, 90, by = 10),
 b = seq(10, 90, by = 10)), show.names = TRUE,
 show.lines = TRUE, col.names = "gray", bg.names = par("bg"),
 show.grid = FALSE, col.axis = "black", col.lines = "gray",
 col.grid = "gray", lty.grid = 3, show.legend = FALSE, label.points = FALSE,
 point.labels = NULL, col.symbols = "black", pch = par("pch"),
 h1 = NA, h3 = NA, t1 = NA, t3 = NA, lwduk = 2, xpos = NA, ypos = NA,
 snames = NA, cexuk = 1.1, ...)
```

Arguments

`soiltexture`	Matrix of soil textures where each row is a soil sample and three columns containing the percentages of the components sand, silt and clay in the range 0 to 100.
`main`	The title of the soil texture plot. Defaults to nothing.

`at`	Positions on the three axes where ticks will be drawn.
`axis.labels`	Labels for the axes.
`tick.labels`	The tick labels for the three axes.
`show.names`	Logical - whether to show the names of different soil types within the soil triangle.
`show.lines`	Logical - whether to show the boundaries of the different soil types within the soil triangle.
`col.names`	Color of the soil names. Defaults to gray.
`bg.names`	Color to use when drawing a blank patch for the names of soil types.
`show.grid`	Logical - whether to show grid lines at each 10 level of each soil component.
`col.axis`	Color of the triangular axes, ticks and labels.
`col.lines`	Color of the boundary lines. Defaults to gray.
`col.grid`	Color of the grid lines. Defaults to gray.
`lty.grid`	Type of line for the grid. Defaults to dashed.
`show.legend`	Logical - whether to display a legend.
`label.points`	Logical - whether to call '`thigmophobe.labels`' to label the points.
`point.labels`	Optional labels for the points or legend.
`col.symbols`	Color of the symbols representing each value.
`pch`	Symbols to use in plotting values.
`h1,h3,t1,t3`	Points used in drawing boundaries for soil types.
`lwduk`	Line width for the boundaries
`xpos,ypos`	Positions for the soil type labels.
`snames`	Soil type labels.
`cexuk`	Character expansion for the soil type labels.
`...`	Additional arguments passed to '`triax.points`' and then '`points`'.

Details

'`soil.texture.uk`' displays a triangular plot area on which soil textures defined as proportions of sand, silt and clay can be plotted. It is similar to the '`soil.texture`' function but uses the UK display conventions.

Value

If '`soiltexture`' was included, a list of the '`x,y`' positions of the soil types plotted. If not, nil.

Author(s)

Julian Stander

See Also

'`triax.plot`'

Examples

```
soils.sw.percent<-data.frame(
 Sand=c(67,67,66,67,36,25,24,59,27,9,8,8,20,
 45,50,56,34,29,39,41,94,98,97,93,96,99),
 Silt=c(17,16,9,8,39,48,54,27,46,70,68,68,66,
 34,30,24,48,53,46,48,2,2,2,4,1,1),
 Clay=c(16,17,25,25,25,27,22,14,27,21,24,24,
 14,21,20,20,18,18,15,11,4,0,1,3,3,0))
soils.sw.cols <- c(1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3,
 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6)
soils.sw.names <- c("Ardington","Astrop","Atrim",
 "Banbury","Beacon","Beckfoot")
soil.texture.uk(soils.sw.percent,
 main = "Ternary Diagram for Some Soils from South West England",
 col.lines = "black", col.names = "black", show.grid = TRUE,
 col.grid = "blue", lty.grid = 2,  pch = 16, cex = 1.0,
 col.symbols = soils.sw.cols, h1 = NA, h3 = NA, t1 = NA,
 t3 = NA , lwduk = 2, xpos = NA, ypos = NA,
 snames = NA, cexuk = 1.1)
legend("topleft", legend = soils.sw.names, col = 1:max(soils.sw.cols),
 pch = 16, cex = 1.1, title = "Location", bty = "n")
```

soils *Soil texture data from 125 soils*

Description

A set of 125 soil texture measurements from soils from various parts of the world.

Usage

```
data(soils)
```

Source

T.H. Skaggs, L.M. Arya, P.J. Shouse and B.P. Mohanty (2001) Estimating Particle-Size Distribution from Limited Soil Texture Data. Soil Science Society of America Journal 65:1038-1044.

spread.labels *Spread labels for irregularly spaced values*

Description

Places labels for irregularly spaced values in a regular staggered order

Usage

```
spread.labels(x,y,labels=NULL,ony=NA,offsets=NA,between=FALSE,
 linecol=par("fg"),srt=0,...)
```

Arguments

`x,y`	x and y data values
`labels`	text strings
`ony`	Whether to force the labels to be spread horizontally (FALSE) or vertically (TRUE). Defaults to whichever way the points are most spread out.
`offsets`	How far away from the data points to place the labels. Defaults to one quarter of the plot span for all, staggered on each side.
`between`	Whether to place the labels between two sets of points.
`linecol`	Optional colors for the lines drawn to the points.
`srt`	Rotation of the labels in degrees.
`...`	additional arguments passed to '`text`'.

Details

This function is mainly useful when labeling irregularly spaced data points that are "spread out" along one dimension. It places the labels regularly spaced and staggered on the long dimension of the data, drawing lines from each label to the point it describes.

If '`between`' is TRUE, the function expects two points for each label and will attempt to place the labels between two vertical lines of points. Lines will be drawn from the ends of each label to the two corresponding points.

If spreading labels horizontally, the user may wish to rotate the labels by 90 degrees ('`srt=90`'). If long labels run off the edge of the plot, increase the '`xlim`' for extra room.

Value

nil

Author(s)

Jim Lemon

References

Cooke, L.J. & Wardle, J. (2005) Age and gender differences in children's food preferences. British Journal of Nutrition, 93: 741-746.

See Also

'`text`'

Examples

```
# spread labels out in the x dimension using defaults
x<-sort(rnorm(10))
y<-rnorm(10)/10
plot(x,y,ylim=c(-1,1),type="p")
nums<-c("one","two","three","four","five","six","seven","eight","nine","ten")
spread.labels(x,y,nums)
# food preferences of children by sex (Cooke & Wardle, 2005)
fpkids<-data.frame(Food=c("Fatty/sugary","Fruit","Starchy","Meat",
 "Proc.meat","Eggs","Fish","Dairy","Vegetables"),
 Female=c(4.21,4.22,3.98,3.57,3.55,3.46,3.34,3.26,3.13),
 Male=c(4.35,4.13,4.02,3.9,3.81,3.64,3.45,3.27,2.96))
plot(rep(1,9),fpkids$Female,xlim=c(0.8,2.2),
 ylim=range(c(fpkids$Female,fpkids$Male)),xlab="Sex",xaxt="n",
 ylab="Preference rating",main="Children's food preferences by sex",
 col="red")
axis(1,at=1:2,labels=c("Female","Male"))
points(rep(2,9),fpkids$Male,col="blue",pch=2)
spread.labels(rep(1:2,each=9),c(fpkids$Female,fpkids$Male),
 fpkids$Food,between=TRUE,linecol=c("red","blue"))
```

`spreadout` *Spread out a vector of numbers to a minimum interval.*

Description

Spread out a vector of numbers so that there is a minimum interval between any two numbers when in ascending or descending order.

Usage

```
spreadout(x,mindist)
```

Arguments

`x`	A numeric vector which may contain NAs.
`mindist`	The minimum interval between any two values when in ascending or descending order.

Details

'`spreadout`' starts at or near the middle of the vector and increases the intervals between the ordered values. NAs are preserved. '`spreadout`' first tries to spread groups of values with intervals less than '`mindist`' out neatly away from the mean of the group. If this doesn't entirely succeed, a second pass that forces values away from the middle is performed.

'`spreadout`' is currently used to avoid overplotting of axis tick labels where they may be close together.

Value

On success, the spread out values. If there are less than two valid values, the original vector is returned.

Author(s)

Jim Lemon

Examples

```
spreadout(c(1,3,3,3,3,5),0.2)
spreadout(c(1,2.5,2.5,3.5,3.5,5),0.2)
spreadout(c(5,2.5,2.5,NA,3.5,1,3.5,NA),0.2)
# this will almost always invoke the brute force second pass
spreadout(rnorm(10),0.5)
```

`stackpoly` *Display the columns of a matrix or data frame as stacked polygons.*

Description

Plot one or more columns of numeric values as the top edges of polygons instead of lines.

Usage

```
stackpoly(x,y=NULL,main="",xlab="",ylab="",xat=NA,xaxlab=NA,
 xlim=NA,ylim=NA,lty=1,lwd=1,border=NA,col=NA,staxx=FALSE,stack=FALSE,
 axis4=TRUE,...)
```

Arguments

`x`	A numeric data frame or matrix with the 'x' values. If 'y' is NULL, these will become the 'y' values and the 'x' positions will be the integers from 1 to dim(x)[1].
`y`	The 'y' values.
`main`	The title for the plot.
`xlab,ylab`	x and y axis labels for the plot.
`xat`	Where to put the optional xaxlabs.
`xaxlab`	Optional labels for the x positions.
`xlim`	Optional x limits.
`ylim`	Optional y limits.
`lty`	Line type for the polygon borders.
`lwd`	Line width for the polygon borders.
`border`	Color for the polygon borders.

`col`	Color to fill the polygons.
`staxx`	Whether to call '`staxlab`' to stagger the x axis labels.
`stack`	Whether to stack the successive values on top of each other.
`axis4`	Whether to display the right ordinate on the plot.
`...`	Additional arguments passed to '`plot`'.

Details

'`stackpoly`' is similar to a line plot with the area under the lines filled with color(s). Ideally, each successive set of y values is greater than the values in the previous set so that the polygons form a rising series of crests. If '`stack`' is TRUE, this is not a problem unless some values of '`x`' are negative.

If '`x`' or '`y`' is a vector, not a matrix or list, the values will be displayed as a "waterfall plot".

Value

nil

Author(s)

Jim Lemon and Thomas Petzoldt (waterfall plot option)

See Also

'`polygon`'

Examples

```
testx<-matrix(abs(rnorm(100)),nrow=10)
stackpoly(matrix(cumsum(testx),nrow=10),main="Test Stackpoly I",
 xaxlab=c("One","Two","Three","Four","Five",
 "Six","Seven","Eight","Nine","Ten"),border="black",staxx=TRUE)
stackpoly(testx,main="Test Stackpoly II",
 xaxlab=c("One","Two","Three","Four","Five",
 "Six","Seven","Eight","Nine","Ten"),border="black",
 staxx=TRUE,stack=TRUE)
stackpoly(rev(sort(testx-mean(testx))),main="Test Waterfall Plot",
 xsrt=45,lwd=3,col="green",border="black")
```

`staircase.plot` *Display a staircase plot.*

Description

Displays a plot showing a sequence of changing totals and increments as successive linked bars.

Usage

```
staircase.plot(heights,totals=NA,labels=NULL,halfwidth=0.3,main="",
mar=NA,total.col="blue",inc.col=NA,bg.col=NA,direction="e",las=1,
display.height=TRUE,stagger=FALSE,...)
```

Arguments

`heights`	vector of numeric values or a matrix or data frame with at least two columns. The first column must be numeric and the second may be numeric or logical.
`totals`	A vector of logicals or zero/non-zero values indicating whether the corresponding height is a total (TRUE) or an increment (FALSE).
`labels`	An optional vector of labels for the bars.
`halfwidth`	Half of the width of a bar as a proportion. See Details.
`main`	A title for the plot.
`mar`	Margins for the plot. Defaults to 10 on the baseline axis, 3 on the top and 1 on the other two sides.
`total.col`	Color(s) for the bars representing successive totals.
`inc.col`	Color(s) for the bars representing increments.
`bg.col`	The background color for the plot.
`direction`	Direction in which the bars should be presented. See Details.
`las`	Orientation for the bar labels. See '`par`'.
`display.height`	Whether to display the totals and increments at the upper ends of the bars. Defaults to TRUE.
`stagger`	Whether to stagger the labels to avoid overlap.
`...`	arguments passed to '`plot`'.

Details

Displays a plot representing successive changes in counts or values. For example, if a research study attempts to contact a certain number of people and some cannot be contacted, some decline to participate, some are ineligible, the final sample will be smaller than the initial contact list. The first value will be the total of attempts, there will be a number of decrements, and the last value will be the actual sample. There may be intermediate totals specified. This produces a visual display of the sampling procedure. See the example.

The bars are placed at integer values on the axis representing the succession of counts or values. The width of the bars is determined by the argument '`halfwidth`'. This defaults to 0.3, meaning that the bar extends 0.3 to each side, so that the proportion of bar to space is 0.6 to 0.4. The succession of bars is determined by the '`direction`' argument. The default is "e" (east), meaning that the first bar is at the left of the plot and subsequent bars are placed to the right. The other three possibilities follow the conventional compass layout.

The '`getFigCtr`' function is called to center the plot title in the figure region as the plot area is typically off center.

Value

nil

Author(s)

Jim Lemon

See Also

'plot', 'getFigCtr'

Examples

```
sample_size<-c(500,-72,428,-94,334,-45,289)
totals<-c(TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE)
labels<-c("Contact list","Uncontactable","","Declined","","Ineligible",
 "Final sample")
staircase.plot(sample_size,totals,labels,main="Acquisition of the sample",
 total.col="gray",inc.col=2:4,bg.col="#eeeebb",direction="s")
```

---

staxlab *Place staggered labels on an axis*

Description

Places labels on an axis in a regular staggered order

Usage

```
staxlab(side=1,at,labels,nlines=2,top.line=0.5,line.spacing=0.8,...)
```

Arguments

`side`	axis on which to place the labels, as in 'axis'
`at`	where to place the labels in user units, as in 'axis'
`labels`	text strings
`nlines`	How many lines to use to stagger the labels.
`top.line`	Distance from the axis to place the first line of text.
`line.spacing`	Spacing between lines of text labels.
`...`	Additional arguments to be passed to 'mtext'.

Value

nil

Note

This function is mainly useful when either long axis labels or a large number of labels are to be placed without overlapping. It staggers the labels along the axis specified. The user may wish to increase the space beneath the plot using ‘`mar`’ before calling ‘`staxlab`’. It is probably only useful on the bottom or left side of the plot.

Author(s)

Jim Lemon

See Also

‘`mtext`’

Examples

```
plot(rnorm(12),axes=FALSE)
box()
months<-c("January","February","March","April","May","June",
 "July","August","September","October","November","December")
staxlab(1,1:12,months)
```

`std.error` *Calculate standard error of the mean*

Description

Calculates the standard error of the mean.

Usage

```
std.error(x,na.rm)
```

Arguments

`x`	A vector of numerical observations.
`na.rm`	Dummy argument to match other functions.

Details

‘`std.error`’ will accept a numeric vector.

Value

The conventional standard error of the mean = sd(x)/sqrt(sum(!is.na(x)))

Author(s)

Jim Lemon

See Also

'`sd`'

`sumDendrite` *Sum the counts in the top level of a dendrite object.*

Description

Find the sum of the counts that are the first elements of each list in the top level of a dendrite object.

Usage

```
sumDendrite(x)
```

Arguments

`x`	A list with a numeric value as the first element in each of its toplevel elements.

Details

A '`dendrite`' object is a possibly nested list of lists that contain the counts and pointers to sublists in each list. Such an object describes the attributes of objects that can take on mutually exclusive attributes (that is, belong to disjunct sets). '`sumDendrite`' is a convenience function to get the total number of objects that are so classified.

Value

The sum of the counts in the top level of lists.

Author(s)

Jim Lemon

See Also

'`plot.dendrite`'

`symbolbarplot` *barplot filled with symbols*

Description

Produces a barplot where each piece of the barplot is filled with the number of symbols equal to the size of the bar

Usage

```
symbolbarplot(height,width=1,space=NULL,names.arg=NULL,
  legend.text=NULL,beside=FALSE,horiz=FALSE,col=heat.colors(NR),
  border=par("fg"),main=NULL,sub=NULL,xlab=NULL,ylab=NULL,xlim=NULL,
  ylim=NULL,axes=TRUE,axisnames=TRUE,inside=TRUE,plot=TRUE,rel.width=0.8,
  symbol="circles",symbbox=TRUE,debug=FALSE,...)
```

Arguments

`height`	numeric vector or matrix of barplot heights
`width`	width of bars
`space`	space between bars
`names.arg`	vector of names
`legend.text`	vector of legend text
`beside`	(logical) plot bars beside each other?
`horiz`	(logical) horizontal barplot?
`col`	vector of colors
`border`	plot border?
`main`	main title
`sub`	subtitle
`xlab`	x axis label
`ylab`	y axis label
`xlim`	x limits
`ylim`	y limits
`axes`	draw axes?
`axisnames`	label horizontal axis?
`inside`	draw lines dividing adjacent bars?
`plot`	produce plot?
`rel.width`	relative width of symbols
`symbol`	which symbol to use
`symbbox`	draw boxes for symbol boxes?
`debug`	debug output?
`...`	further arguments to multsymbolbox

Value

Nil

Note

This is a mostly a hack of barplot()

Author(s)

Ben Bolker

Examples

```
set.seed(1001)
bvals <- matrix(rpois(12,20),nrow=3)
b <- symbolbarplot(bvals)
```

---

symbolbox *Draw a box filled with symbols*

Description

Draws a box on the current figure that is filled with symbols representing individual counts

Usage

```
symbolbox(x1,y1,x2,y2,tot,relw=0.5,fg=par("fg"),bg=par("bg"),box=TRUE,
 debug = TRUE,...)
```

Arguments

`x1`	left side of box
`y1`	bottom side of box
`x2`	right side of box
`y2`	top side of box
`tot`	total number of symbols to put in the box
`relw`	relative width (relative to height) of symbols
`fg`	foreground color
`bg`	background color
`box`	(logical) draw box border?
`debug`	debug output?
`...`	additional arguments to polygon() for drawing box

Details

tries to automatically figure out appropriate scaling to fit symbols into the box

Value

none; draws on the current figure

Author(s)

Ben Bolker

See Also

'`multsymbolbox`'

Examples

```
plot(1:10,1:10,type="n")
symbolbox(2,5,3,7,tot=20)
symbolbox(6,2,10,6,tot=50,fg="blue",bg="magenta")
```

`tab.title` *Display the title of a plot as a colored tab.*

Description

Display the title of a plot as a colored tab.

Usage

```
tab.title(label,text.col=par("fg"),tab.col=par("bg"),border=par("fg"),
  lwd=par("lwd"),cex=1.5,pad.mult=1.6,radius=0)
```

Arguments

`label`	The title for the plot.
`text.col`	The color for the title text.
`tab.col`	The color for the tab fill.
`border`	The color for the tab border.
`lwd`	The line width for the border.
`cex`	Character expansion for the title.
`pad.mult`	How much higher to make the tab relative to the label.
`radius`	What proportion of the tab corners to round off.

Details

'`tab.title`' displays the plot title in a colored tab. The tab can be rounded at the upper corners by specifying the proportion of the tab height to be rounded as a number between 0 and 1. If the tab is too high to fit on the figure region, a warning will be displayed and the tab will still be shown.

Value

nil

Author(s)

Jim Lemon

See Also

'`polygon`'

Examples

```
testx<-matrix(cumsum(rnorm(30)^2)+1,nrow=10)
stackpoly(testx,main="",
 xaxlab=c("One","Two","Three","Four","Five",
 "Six","Seven","Eight","Nine","Ten"),staxx=TRUE)
tab.title("Three Squiggly Lines",tab.col="yellow",radius=0.5)
```

`taylor.diagram` *Taylor diagram*

Description

Display a Taylor diagram.

Usage

```
taylor.diagram(ref,model,add=FALSE,col="red",pch=19,pos.cor=TRUE,
 xlab="",ylab="",main="Taylor Diagram",show.gamma=TRUE,ngamma=3,
 sd.arcs=0,ref.sd=FALSE,grad.corr.lines=c(0.2,0.4,0.6,0.8,0.9),
 pcex=1,normalize=FALSE,...)
```

Arguments

`ref`	numeric vector - the reference values.
`model`	numeric vector - the predicted model values.
`add`	whether to draw the diagram or just add a point.
`col`	the color for the points displayed.
`pch`	the type of point to display.

`pos.cor`	whether to display only positive (‘`TRUE`’) or all values of correlation (‘`FALSE`’).
`xlab,ylab`	plot axis labels.
`main`	title for the plot.
`show.gamma`	whether to display standard deviation arcs around the reference point (only for ‘`pos.cor=TRUE`’).
`ngamma`	the number of gammas to display (default=3).
`sd.arcs`	whether to display arcs along the standard deviation axes (see Details).
`ref.sd`	whether to display the arc representing the reference standard deviation.
`grad.corr.lines`	the values for the radial lines for correlation values (see Details).
`pcex`	character expansion for the plotted points.
`normalize`	whether to normalize the models so that the reference has a standard deviation of 1.
`...`	Additional arguments passed to ‘`plot`’.

Details

The Taylor diagram is used to display the quality of model predictions against the reference values, typically direct observations.

A diagram is built by plotting one model against the reference, then adding alternative model points. If ‘`normalize=TRUE`’ when plotting the first model, remember to set it to ‘`TRUE`’ when plotting additional models.

Two displays are available. One displays the entire range of correlations from -1 to 1. Setting ‘`pos.cor`’ to ‘`FALSE`’ will produce this display. The -1 to 1 display includes a radial grid for the correlation values. When ‘`pos.cor`’ is set to ‘`TRUE`’, only the range from 0 to 1 will be displayed. The ‘`gamma`’ lines and the arc at the reference standard deviation are optional in this display.

Both the standard deviation arcs and the gamma lines are optional in the ‘`pos.cor=TRUE`’ version. Setting ‘`sd.arcs`’ or ‘`grad.corr.lines`’ to zero or FALSE will cause them not to be displayed. If more than one value is passed for ‘`sd.arcs`’, the function will try to use the values passed, otherwise it will call ‘`pretty`’ to calculate the values.

Value

The values of ‘`par`’ that preceded the function. This allows the user to add points to the diagram, then restore the original values. This is only necessary when using the 0 to 1 correlation range.

Author(s)

Olivier Eterradossi with modifications by Jim Lemon

References

Taylor, K.E. (2001) Summarizing multiple aspects of model performance in a single diagram. Journal of Geophysical Research, 106: 7183-7192.

Examples

```
# fake some reference data
ref<-rnorm(30,sd=2)
# add a little noise
model1<-ref+rnorm(30)/2
# add more noise
model2<-ref+rnorm(30)
# display the diagram with the better model
oldpar<-taylor.diagram(ref,model1)
# now add the worse model
taylor.diagram(ref,model2,add=TRUE,col="blue")
# get approximate legend position
lpos<-1.5*sd(ref)
# add a legend
legend(lpos,lpos,legend=c("Better","Worse"),pch=19,col=c("red","blue"))
# now restore par values
par(oldpar)
# show the "all correlation" display
taylor.diagram(ref,model1,pos.cor=FALSE)
taylor.diagram(ref,model2,add=TRUE,col="blue")
```

`textbox` *Add text box*

Description

Add text to plot, justified, in a box

Usage

```
textbox(x,y,textlist,justify=TRUE,cex=1,leading=0.5,box=TRUE)
```

Arguments

`x`	x position: a vector with min. and max. x position
`y`	y position: location of the top of the box
`textlist`	a vector of text strings
`justify`	justify text in box?
`cex`	character size
`leading`	inter-line spacing
`box`	draw a box around the text?

Details

Justifies text in the box by pasting the vector together, splitting it into words, and then adding words to the current line until the line is wide enough

Value

nil

Author(s)

Ben Bolker

Examples

```
plot.new()
textbox(c(0,0.2),1,c("many words","more words","why not?",
                     "keep going",rep("and going",10)))
```

`thigmophobe` *Find the direction away from the closest point*

Description

Find the direction away from the closest point

Usage

```
thigmophobe(x,y)
```

Arguments

`x,y`	Numeric data vectors. Typically the x/y coordinates of plotted points. If arrays are passed, they will be silently coerced to numeric vectors.

Details

'`thigmophobe`' returns the direction (as 1|2|3|4 - see pos= in '`text`') away from the nearest point to each of the points described by '`x`' and '`y`'.

Value

A vector of directions away from the point nearest to each point.

Note

Typically used to get the offset to automatically place labels on a scatterplot or similar using '`thigmophobe.labels`' to avoid overlapping labels. The name means "one who fears being touched".

Author(s)

Jim Lemon - thanks to Gustaf Rydevik for the "names" bug fix and to Steve Ellison for the suggestion about arrays.

See Also

'thigmophobe.labels'

Examples

```
x<-rnorm(10)
y<-rnorm(10)
thigmophobe(x,y)
```

---

thigmophobe.labels *Place labels away from the nearest point*

Description

'thigmophobe.labels' places labels adjacent to each point, offsetting each label in the direction returned by 'thigmophobe'.

Usage

```
thigmophobe.labels(x,y,labels=NULL,text.pos=NULL,...)
```

Arguments

`x,y`	Numeric data vectors or a list with two components. Typically the x/y coordinates of plotted points.
`labels`	A vector of strings that will be placed adjacent to each point. Defaults to the indices of the coordinates.
`text.pos`	An optional vector of text positions (see 'text').
`...`	additional arguments are passed to 'text'

.

Details

Typically used to automatically place labels on a scatterplot or similar to avoid overlapping labels. 'thigmophobe.labels' will sometimes place a label off the plot or fail to separate labels in clusters of points. The user can manually adjust the errant labels by running 'thigmophobe' first and saving the returned vector. Then modify the position values to place the labels properly and pass the edited vector to 'thigmophobe.labels' as the 'text.pos' argument. This takes precedence over the positions calculated by 'thigmophobe'.

Both 'pointLabel' in the **maptools** package and 'spread.labs' in the **TeachingDemos** package use more sophisticated algorithms to place the labels and are worth a try if 'thigmophobe' just won't get it right.

Value

A vector of directions away from the point nearest to each point.

Author(s)

Jim Lemon

See Also

'thigmophobe', 'text'

Examples

```
x<-rnorm(20)
y<-rnorm(20)
xlim<-range(x)
xspace<-(xlim[2]-xlim[1])/20
xlim<-c(xlim[1]-xspace,xlim[2]+xspace)
ylim<-range(y)
yspace<-(ylim[2]-ylim[1])/20
ylim<-c(ylim[1]-yspace,ylim[2]+yspace)
plotlabels<-
 c("one","two","three","four","five","six","seven","eight","nine","ten",
 "eleven","twelve","thirteen","fourteen","fifteen","sixteen","seventeen",
 "eighteen","nineteen","twenty")
plot(x=x,y=y,xlim=xlim,ylim=ylim,main="Test thigmophobe.labels")
# skip the almost invisible yellow label, make them bold and without borders
thigmophobe.labels(x,y,plotlabels,col=c(2:6,8:12),border=NA,font=2)
```

`triax.abline` *Lines for triangle plot*

Description

Display lines on a triangle plot.

Usage

```
triax.abline(b=NULL,r=NULL,l=NULL,col=par("col"),lty=par("lty"),
  cc.axes=FALSE)
```

Arguments

`b`	Lines relating to the bottom axis.
`r`	Lines relating to the right axis.
`l`	Lines relating to the left axis.
`col`	Color(s) of the lines.
`lty`	Type(s) of the lines.
`cc.axes`	Clockwise/counterclockwise axes and ticks.

Details

'triax.abline' displays one or more lines on a triangle plot. Lines are oriented in the conventional way, horizontal for the left axis, slanting up to the right for the right axis and up to the left for the bottom axis. If 'cc.axes' is TRUE, the orientation is up-left for the left axis, horizontal for the right axis and up-right for the bottom axis.

Remember to call 'triax.plot' with 'no.add=FALSE' and restore the graphics parameters as in the example or the lines will not be placed properly.

Value

nil

Author(s)

Jim Lemon

See Also

'triax.plot'

Examples

```
triax.return<-triax.plot(data.frame(bottom=0.4,right=0.3,left=0.3),
 main="Triax ablines",no.add=FALSE)
triax.abline(l=0.3,col="red")
triax.abline(r=0.3,col="green")
triax.abline(b=0.4,col="blue")
par(triax.return$oldpar)
```

---

triax.frame *Triangle plot frame*

Description

Display a three axis frame with optional grid.

Usage

```
triax.frame(main="",at=seq(0.1,0.9,by=0.1),
axis.labels=NULL,tick.labels=NULL,col.axis="black",cex.axis=1,cex.ticks=1,
align.labels=TRUE,show.grid=FALSE,col.grid="gray",lty.grid=par("lty"),
cc.axes=FALSE)
```

Arguments

`main`	The title of the triangle plot. Defaults to nothing.
`at`	The tick positions on the three axes.
`axis.labels`	Labels for the three axes in the order bottom, right left. Defaults to the column names.
`tick.labels`	The tick labels for the axes. Defaults to argument '`at`' (proportions).
`col.axis`	Color of the triangular axes, ticks and labels.
`cex.axis`	Character expansion for axis labels.
`cex.ticks`	Character expansion for the tick labels.
`align.labels`	Logical - whether to align axis and tick labels with the axes.
`show.grid`	Whether to display grid lines at the ticks.
`col.grid`	Color of the grid lines. Defaults to gray.
`lty.grid`	Type of line for the grid.
`cc.axes`	Whether to align the axes clockwise or counterclockwise.

Details

'`triax.frame`' displays a triangular plot area on which proportions or percentages may be displayed. An optional grid may also be displayed. If '`cc.axes`' is TRUE, both the axes and axis ticks will be in reverse order.

Value

nil

Author(s)

Jim Lemon

See Also

'`triax.points`','`triax.abline`'

Examples

```
triax.frame(main="DEFAULT")
triax.frame(main="Clockwise axes",cc.axes=TRUE)
```

triax.plot *Triangle plot*

Description

Display a triangle plot with optional grid.

Usage

```
triax.plot(x=NULL,main="",at=seq(0.1,0.9,by=0.1),
axis.labels=NULL,tick.labels=NULL,col.axis="black",cex.axis=1,cex.ticks=1,
align.labels=TRUE,show.grid=FALSE,col.grid="gray",lty.grid=par("lty"),
cc.axes=FALSE,show.legend=FALSE,label.points=FALSE,point.labels=NULL,
col.symbols="black",pch=par("pch"),no.add=TRUE,...)
```

Arguments

`x`	Matrix where each row is three proportions or percentages that must sum to 1 or 100 respectively.
`main`	The title of the triangle plot. Defaults to nothing.
`at`	The tick positions on the three axes.
`axis.labels`	Labels for the three axes in the order left, right, bottom. Defaults to the column names.
`tick.labels`	The tick labels for the three axes as a list with three components l, r and b (left, right and bottom). Defaults to argument '`at`' (proportions).
`col.axis`	Color of the triangular axes, ticks and labels.
`cex.axis`	Character expansion for axis labels.
`cex.ticks`	Character expansion for the tick labels.
`align.labels`	Logical - whether to align axis and tick labels with the axes.
`show.grid`	Whether to display grid lines at the ticks.
`col.grid`	Color of the grid lines. Defaults to gray.
`lty.grid`	Type of line for the grid.
`cc.axes`	Whether axes and axis ticks should be clockwise or counterclockwise.
`show.legend`	Logical - whether to display a legend.
`label.points`	Logical - whether to call '`thigmophobe.labels`' to label the points.
`point.labels`	Optional labels for the points and/or legend.
`col.symbols`	Color of the symbols representing each value.
`pch`	Symbols to use in plotting values.
`no.add`	Whether to restore the previous plotting parameters ('`TRUE`') or leave them, allowing more points to be added.
`...`	Additional arguments passed to '`points`'.

Details

'triax.plot' displays a triangular plot area on which proportions or percentages are displayed. A grid or legend may also be displayed.

Value

A list containing 'xypos' (the 'x,y' positions plotted) and 'oldpar' (the plotting parameters at the time 'triax.plot' was called).

Note

A three axis plot can only properly display one or more sets of three proportions that each sum to 1 (or percentages that sum to 100). Other values may be scaled to proportions (or percentages), but unless each set of three sums to 1 (or 100), they will not plot properly and 'triax.points' will complain appropriately. Note also that 'triax.plot' will only display properly in a square plot, which is forced by 'par(pty="s")'.

In case the user does want to plot values with different sums, the axis tick labels can be set to different ranges to accomodate this. 'triax.points' will still complain, but it will plot the values.

If planning to add points with 'triax.points' call 'triax.plot' with 'no.add=FALSE' and restore plotting parameters after the points are added.

Author(s)

Jim Lemon - thanks to Ben Daughtry for the info on counterclockwise axes.

See Also

'triax.points', 'triax.abline', 'thigmophobe.labels'

Examples

```
data(soils)
triax.plot(soils[1:10,],main="DEFAULT")
triax.plot(soils[1:10,],main="PERCENTAGES (Counterclockwise axes)",
 tick.labels=list(l=seq(10,90,by=10),r=seq(10,90,by=10),b=seq(10,90,by=10)),
 pch=3,cc.axes=TRUE)
triax.return<-triax.plot(soils[1:6,],main="GRID AND LEGEND",
 show.grid=TRUE,show.legend=TRUE,col.symbols=1:6,pch=4)
# triax.plot changes a few parameters
par(triax.return$oldpar)
```

triax.points *Triangle plot points*

Description

Display points on a triangle plot.

Usage

```
triax.points(x,show.legend=FALSE,label.points=FALSE,point.labels=NULL,
col.symbols=par("fg"),pch=par("pch"),bg.symbols=par("bg"),cc.axes=FALSE,...)
```

Arguments

`x`	Matrix or data frame where each row is three proportions or percentages that must sum to 1 or 100 respectively.
`show.legend`	Logical - whether to display a legend.
`label.points`	Logical - whether to call '`thigmophobe.labels`' to label the points.
`point.labels`	Optional labels for the points and/or legend.
`col.symbols`	Color of the symbols representing each value.
`pch`	Symbols to use in plotting values.
`bg.symbols`	Background color for plotting symbols.
`cc.axes`	Clockwise or counterclockwise axes and ticks.
`...`	Additional arguments passed to '`points`'.

Details

In order for '`triax.points`' to add points to an existing plot, the argument '`no.add`' in the initial call to '`triax.plot`' must be set to '`FALSE`'. Failing to do this will result in the points being plotted in the wrong places. It is then up to the user to call '`par`' as in the example below to restore plotting parameters altered during the triangle plot.

'`triax.points`' displays each triplet of proportions or percentages as a symbol on the triangle plot. Unless each triplet sums to 1 (or 100), they will not plot properly and '`triax.points`' will complain appropriately.

Value

A list of the '`x,y`' positions plotted.

Author(s)

Jim Lemon

See Also

'`triax.plot`','`thigmophobe.labels`'

Examples

```
data(soils)
triax.return<-triax.plot(soils[1:10,],
 main="Adding points to a triangle plot",no.add=FALSE)
triax.points(soils[11:20,],col.symbols="green",pch=3)
par(triax.return$oldpar)
```

`twoord.plot` *Plot with two ordinates*

Description

Two sets of values are displayed on the same plot with different ordinate scales on the left and right.

Usage

```
twoord.plot(lx,ly,rx,ry,data=NULL,xlim=NULL,lylim=NULL,rylim=NULL,
mar=c(5,4,4,4),lcol=1,rcol=2,xlab="",ylab="",rylab="",lpch=1,rpch=2,
type="b",xtickpos=NULL,xticklab=NULL,halfwidth=0.4,axislab.cex=1,...)
```

Arguments

`lx,ly,rx,ry`	y and optional x values for the plot
`data`	an optional data frame from which to obtain the above values
`xlim`	optional x limits as in ‘`plot`’
`lylim,rylim`	optional y limits for the left and right axes respectively
`mar`	optional margin adjustment, defaults to ‘`c(5,4,4,4)`’
`lcol,rcol`	colors to distinguish the two sets of values
`xlab,ylab`	axis labels as in ‘`plot`’
`rylab`	label for the right axis
`lpch,rpch`	plot symbols to distinguish the two sets of values
`type`	as in ‘`plot`’
`xtickpos`	Optional positions for x-axis tick labels.
`xticklab`	Optional labels for x-axis. Useful for things like dates.
`halfwidth`	Half the width of the bars in user units. The bars are centered on successive integers if no ‘`x`’ values are supplied.
`axislab.cex`	Character expansion for the axis labels and tick labels.
`...`	additional arguments passed to ‘`plot`’.

Details

'`twoord.plot`' automates the process of displaying two sets of values that have different ranges on the same plot. It is principally useful in illustrating some relationship between the values across the observations. It is assumed that the '`lx`' and '`rx`' values are at least adjacent, and probably overlapping.

It is best to pass all the arguments '`lx,ly,rx,ry`', but the function will attempt to substitute sensible x values if one or two are missing.

If at least one of the '`type`' arguments is "bar", bars will be plotted instead of points or lines. It is best to plot the bars first (i.e. relative to the left axis) if the other type is points or lines, as the bars will usually obscure at least some of the points or lines. Using NA for the color of the bars will partially correct this. If both types are to be bars, remember to pass somewhat different x values or the bars will be overlaid.

Value

nil

Note

There are many objections to the use of plots with two different ordinate scales, and some of them are even sensible and supported by controlled observation. Many of the objections rest on assertions that the spatial arrangement of the values plotted will override all other evidence. Here are two:

The viewer will assume that the vertical position of the data points indicates a quantitative relationship.

To some extent. It is probably not a good idea to have the spatial relationship of the points opposed to their numerical relationship. That is to say, if one set of values is in the range of 0-10 and the other 20-100, it is best to arrange the plot so that the latter values are not plotted below the former.

The viewer will assume that an intersection of lines indicates an intersection of values.

If the visual elements representing values can be arranged to avoid intersections, so much the better. Many people have no trouble distinguishing which visual elements are linked to which axis as long as they are both coded similarly, usually with colors and/or symbols. In the special case where there is an underlying relationship between the two such as the probability of that value occurring under some conditions, it may help to mark the point(s) where this occurs.

It may be useful to consider '`gap.plot`' as an alternative.

Author(s)

Jim Lemon (thanks to Christophe Dutang for the idea of using bars and lines in the same plot, Clair Crossupton for pointing out that dates on the x-axis weren't very good and Jacob Kasper for the axis character expansion.)

See Also

'`plot`'

Examples

```
twoord.plot(2:10,seq(3,7,by=0.5)+rnorm(9),
 1:15,rev(60:74)+rnorm(15),xlab="Sequence",
 ylab="Ascending values",rylab="Descending values",
 main="Plot with two ordinates - points and lines")
twoord.plot(2:10,seq(3,7,by=0.5)+rnorm(9),
 1:15,rev(60:74)+rnorm(15),xlab="Sequence",
 ylab="Ascending values",rylab="Descending values",
 main="Plot with two ordinates - bars on the left",
 type=c("bar","l"),lcol=3,rcol=4)
twoord.plot(2:10,seq(3,7,by=0.5)+rnorm(9),
 1:15,rev(60:74)+rnorm(15),xlab="Sequence",
 ylab="Ascending values",rylab="Descending values",
 main="Plot with two ordinates - bars on the right",
 type=c("b","bar"),lcol=2,rcol=NA,halfwidth=0.2)
# histogram with density curve overlaid
xhist<-hist(rnorm(100),plot=FALSE)
xdens<-dnorm(seq(-2,2,by=0.05))
twoord.plot(xhist$mids,xhist$counts,seq(-2,2,by=0.05),
xdens,type=c("bar","l"),lcol=4,rcol=2,ylab="Counts",
rylab="Density",main="Histogram and density curve",
halfwidth=0.2)
```

`vectorField` *Display magnitude/direction vectors*

Description

Display magnitude/direction vectors as arrows on an existing plot.

Usage

```
vectorField(u,v,xpos=NA,ypos=NA,scale=1,headspan=0.1,
 vecspec=c("lonlat","rad","deg"))
```

Arguments

`u,v`	x (longitude) and y (latitude) offsets OR orientation and magnitude in either radians or degrees. See details.
`xpos,ypos`	The centers of the vectors in user units.
`scale`	The proportion of each cell that the maximal vector will fill. See details.
`headspan`	The extent of the heads of the arrows as a proportion of cell size.
`vecspec`	How the vectors are described. See details

Details

'vectorField' displays arrows on an existing plot. Each arrow is specified by a position on the plot 'xpos,ypos' and either x/y offsets or orientation and magnitude. The default is x/y offsets, and the user must specify whether radians or degrees are used if the orientation/magnitude option is used.

If the first four arguments are matrices, there must be no missing values. If these arguments are vectors, the calculation of the scaling of the magnitudes and length of the arrowheads may be slightly different.

Value

nil

Author(s)

Jim Lemon (original code by Robin Hankin and Brian Ripley)

See Also

'arrows'

Examples

```
 ## Not run:
  # this requires the maps package, and just wouldn't pass check
  require(maps)
  map("world",xlim=c(110,155),ylim=c(-40,-10))
  par(xpd=TRUE)
  text(132,-5,"Approximate magnetic deviation - Australia",cex=1.5)
  par(xpd=FALSE)
  long<-rep(seq(117.5,152.5,by=5),6)
  lat<-rep(c(-12.5,-17.5,-22.5,-27.5,-32.5,-37.5),each=8)
  # just show the direction, don't have a magnitude difference
  mag<-rep(1,48)
  devdeg<-c(110,98,85,65,65,65,65,65,
   115,100,90,80,72,66,63,55,
   130,100,90,82,72,67,62,54,
   122,111,95,86,70,67,56,48,
   118,116,110,87,74,68,62,45,
   128,115,107,90,78,66,53,45)
  vectorField(devdeg,mag,long,lat,scale=0.7,vecspec="deg")

## End(Not run)
 # do a magnitude/direction plot with radians
 plot(1:10,type="n",main="Random vectors")
 mag<-runif(100)+1
 dir<-runif(100)*2*pi
 xpos<-rep(1:10,10)
 ypos<-rep(1:10,each=10)
 vectorField(dir,mag,xpos,ypos,scale=0.8,vecspec="rad")
```

`weighted.hist` *Display a weighted histogram*

Description

Calculate the counts of the weighted values in specified bins and optionally display either a frequency or density histogram.

Usage

```
weighted.hist(x,w,breaks="Sturges",col=NULL,plot=TRUE,
freq=TRUE,ylim,ylab=NULL,...)
```

Arguments

`x`	A vector of numeric values
`w`	A vector of weights at least as long as x.
`breaks`	The endpoints of the ranges into which to count the weighted values.
`col`	An optional vector of colors for the bars of the histogram.
`plot`	Whether to plot a histogram.
`freq`	Whether to plot counts or densities.
`ylim`	The limits of the plot ordinate.
`ylab`	Label for the ordinate.
`...`	Additional arguments passed to '`barplot`'.

Details

'`weighted.hist`' calculates the weighted counts of values falling into the ranges specified by '`breaks`'. Instead of counting each value as 1, it counts the corresponding value in '`w`' (the weight).

'`breaks`' may be specified by a monotonically increasing vector of numbers that are interpreted as the endpoints of the ranges, a single number representing the number of ranges desired or the name of the function to calculate the ranges (see '`hist`'). If a vector of numbers is passed that does not include all values in '`x`', the user is warned. If the ranges are not equal, a warning will be displayed if '`freq`' is TRUE or the heights of the bars will be adjusted to display areas approximately equal to the counts if '`freq`' is FALSE.

Value

A list containing:

breaks - The endpoints of the intervals

counts - The weighted counts

density - The weighted counts divided by their sum.

mids - The midpoints of the intervals and the bars displayed.

xname - the name of ‘x’.

equidist - Whether the intervals differ by less than the total range/1000.

Author(s)

Jim Lemon and Hadley Wickham

See Also

‘hist’

Examples

```
testx<-sample(1:10,300,TRUE)
testw<-seq(1,4,by=0.01)
weighted.hist(testx,testw,breaks=1:10,main="Test weighted histogram")
```

---

zoomInPlot *Display a plot with a rectangular section expanded in an adjacent plot.*

Description

Display one plot on the left half of a device and an expanded section of that plot on the right half of the device with connecting lines showing the expansion.

Usage

```
zoomInPlot(x,y=NULL,xlim=NULL,ylim=NULL,rxlim=xlim,rylim=ylim,xend=NA,
 zoomtitle=NULL,...)
```

Arguments

`x,y`	numeric data vectors. If ‘y’ is not specified, it is set equal to ‘x’ and ‘x’ is set to ‘1:length(y)’.
`xlim,ylim`	Limits for the initial plot.
`rxlim,rylim`	Limits for the expanded plot. These must be within the above.
`xend`	Where to end the segments that indicate the expansion. Defaults to just left of the tick labels on the left ordinate.
`zoomtitle`	The title of the plot, display in the top center.
`...`	additional arguments passed to ‘plot’.

Details

‘zoomInPlot’ sets up a two column layout in the current device and calls ‘plot’ to display a plot in the left column. It then draws a rectangle corresponding to the ‘rxlim’ and ‘rylim’ arguments and displays a second plot of that rectangle in the right column. It is currently very simple and will probably become more flexible in future versions.

Value

nil

Author(s)

Jim Lemon

See Also

'`plot`'

Examples

```
zoomInPlot(rnorm(100),rnorm(100),rxlim=c(-1,1),rylim=c(-1,1),
 zoomtitle="Zoom In Plot")
```

Index