Package 'plotrix'

September 8, 2021

Version 3.8-2

Date 2021-09-08

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, Rolf Turner,
Carl Witthoft, Julian Stander, Thomas Petzoldt,
Remko Duursma, Elisa Biancotto, Ofir Levy,
Christophe Dutang, Peter Solymos, Robby Engelmann,
Michael Hecker, Felix Steinbeck, Hans Borchers,
Henrik Singmann, Ted Toal, Derek Ogle, Darshan Baral,
Ulrike Groemping, Bill Venables

Maintainer Jim Lemon <drjimlemon@gmail.com>

Imports grDevices, graphics, stats, utils

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

License GPL (>= 2)

NeedsCompilation no

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2021-09-08 11:00:02 UTC

R topics documented:

`plotrix-package` *Specialized plots and plotting accessories*

Description

A large number of specialized plots and accessory functions like color scaling, text placement and legends.

Details

Package:	plotrix
Version:	3.8-1
Date:	2021-09-08
License:	GPL (>=2)
Packaged:	2021-09-08 03:45:00 UTC; root
Built:	R 4.0.3; ; 2021-09-08 03:45:00 UTC; linux

The plotrix package is intended to provide a method for getting many sorts of specialized plots quickly, yet allow easy customization of those plots without learning a great deal of specialized syntax. There are three major aims that can be represented as follows:

Fast foods

Think of plotrix as a graphics vending machine or fast graphics cafe. You walk in, make your choice and get your lunch. It may not be exactly the lunch you want, but you do get a pretty good lunch, fast. You can get junk food or health food, you make the choice.

Hot rods

You can customize plotrix as much as you want. Like the ageing machinery that is usually bolted into hot rods, the base graphics package is fairly easy to understand. plotrix is modular. You can create a frame for your plot, then you can add whatever bits you like to it instead of just taking the default plot that is available. You can have wide wheels and chromed exhaust pipes if you want.

No black boxes

If you want to go from pushing the fast food button to hot rodding, it's not hard. The source code in the functions is written to be understood. If something goes wrong, you can usually find where it happened right away and work on it. This means that you can learn about how the functions do what they do rather than just what they do. So that's how to write recursive functions in R!

Because plotrix encourages users to learn how it works, you usually begin to do so pretty quickly. Users often decide to write their own versions of plotrix functions and sometimes they contribute the results back into plotrix. You may find that you like other graphics systems like grid or lattice better. That's great, because one idea behind plotrix is that if you get into R and can get things done quickly and easily, you'll stick with it and soon want to get things done your way.

Author(s)

Jim Lemon <drjimlemon@gmail.com>, and many others

Maintainer: Jim Lemon <drjimlemon@gmail.com>

`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 y-value(s) for horizontal line(s).
`v`	the x-value(s) for vertical 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.

Multiple lines of the same type can be drawn in a single call, but the clipping region must be the same for each group of lines. Thanks to Berry Boessenkool for pointing this out.

Value

None. Adds to the current plot.

Author(s)

Remko Duursma

See Also

abline, clip

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")
```

---

add.ps *add p-values from t-tests*

Description

Adds p-values comparing the different cells at each x-axis position with a reference cell. Uses a syntax similar to 'raw.means.plot2'.

Usage

```
add.ps(data, col.id, col.offset, col.x, col.value, fun.aggregate = "mean",
ref.offset = 1, prefixes,alternative = c("two.sided", "less", "greater"),
mu = 0, paired = FALSE, var.equal = FALSE, lty = 0, ...)
```

Arguments

data	A 'data.frame'
col.id	'character' vector specifying the id column.
col.offset	'character' vector specifying the offset column.
col.x	'character' vector specifying the x-axis column.
col.value	'character' vector specifying the data column.
fun.aggregate	Function or function name used for aggregating the results. Default is '"mean"'.
ref.offset	Scalar 'numeric' indicating the reference level to be tested against. The default is 1 corresponding to 'levels(factor(d[,col.offset]))[1]'.
prefixes	'character' vector of the indices for the p-values. If missing corresponds to 'levels(factor(d.new[,col.offset]))[-ref.offset]'.
alternative	same as in t.test
mu	same as in t.test
paired	same as in t.test
var.equal	same as in t.test
lty	line type of axis, Default is 0 (i.e., no line).
...	further arguments passed to axis.

Details

This function computes t-tests comparing the values at each x-axis position for each condition against the reference condition at and adds the p-values to the axis.

This functions uses the same syntax as raw.means.plot2 and should be used in addition to it. Note that values are ordered according to the 'col.id' so 'paired = TRUE' should be fine.

Value

axis is plotted.

Author(s)

Henrik Singmann

See Also

raw.means.plot as the accompanying main functions.

Examples

```
## Not run:
#The examples uses the OBrienKaiser dataset from car and needs reshape.
# This extends the examples from raw.means.plot
require(reshape)
require(car)
data(OBrienKaiser)
```

```
OBKnew <- cbind(factor(1:nrow(OBrienKaiser)), OBrienKaiser)
colnames(OBKnew)[1] <- "id"
OBK.long <- melt(OBKnew)
OBK.long[, c("measurement", "time")] <-
 t(vapply(strsplit(as.character(OBK.long$variable), "\\."),  "[", c("", "")))

# For this example the position at each x-axis are within-subject comparisons!
raw.means.plot2(OBK.long, "id", "measurement", "gender", "value")
 add.ps(OBK.long, "id", "measurement", "gender", "value", paired = TRUE)
 #reference is "fup"

raw.means.plot2(OBK.long, "id", "measurement", "gender", "value")
add.ps(OBK.long, "id", "measurement", "gender", "value", ref.offset = 2,
 paired = TRUE) #reference is "post"

# Use R's standard (i.e., Welch test)
raw.means.plot2(OBK.long, "id", "treatment", "gender", "value")
add.ps(OBK.long, "id", "treatment", "gender", "value",
 prefixes = c("p(control vs. A)", "p(control vs. B)"))

# Use standard t-test
raw.means.plot2(OBK.long, "id", "treatment", "gender", "value")
add.ps(OBK.long, "id", "treatment", "gender", "value", var.equal = TRUE,
 prefixes = c("p(control vs. A)", "p(control vs. B)"))


## End(Not run)
```

`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,xpad=0.1,ypad=0.5,box.col=par("fg"),text.col=par("fg"),
 display.colnames=TRUE,display.rownames=FALSE,hlines=FALSE,vlines=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'.
xpad,ypad	The amount of padding around text in the cells as a proportion of the maximum width and height of the strings in each column.
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.
vlines	Whether to draw vertical lines between each column 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 function now accepts the positional arguments such as "topright" if passed as 'x'. The defaults are those that were most popular in scientific journals at the time of programming.

If 'bg' is a matrix of colors of the same dimensions as 'x', those colors will be the backgrounds of the cells. The default is no background color.

Value

nil

Author(s)

Original by John Kane, mods by Jim Lemon and Brian Diggs. Thanks to Andrija Djurovic for asking for the individual cell colors and Gabor Grothendieck for alerting me to the problem of widely varying column widths.

See Also

legend

Examples

```
testdf <- data.frame(Before = c(10, 7, 5, 9), During = c(8, 6, 2, 5),
 After = c(5, 3, 4, 3))
rownames(testdf) <- c("Red", "Green", "Blue", "Lightblue")
barp(testdf, main = "Test addtable2plot", ylab = "Value",
 names.arg = colnames(testdf), col = 2:5)
# show most of the options including the christmas tree colors
```

```
abg <- matrix(c(2, 3, 5, 6, 7, 8), nrow=4, ncol=3)
addtable2plot(2, 8, testdf, bty = "o", display.rownames = TRUE, hlines = TRUE,
 vlines = TRUE, title = "The table", bg = abg)
```

---

arctext *Display text on a circular arc*

Description

Displays a character string on the circumference of an imaginary circle on an existing plot.

Usage

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

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.
end	The end position of the string in radians.
stretch	How much to stretch the string for appearance.
clockwise	Whether to print the string in the clockwise direction.
cex	The character expansion factor.
...	additional arguments passed to 'text'.

Details

'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.

If the 'clockwise' argument is TRUE, the string will be displayed in a clockwise direction and the orientation of the characters will be rotated 'pi' radians (180 degrees). This is useful when the string is to be displayed on the bottom of the circumference.

Value

nil

Author(s)

Jim Lemon - Thanks to Suhas Parandekar for the idea, Ted Toal for greatly improving the placement of the text and Andy South for providing the initial code for the clockwise argument.

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)
arctext("bendy like spaghetti", center = c(3, 3), radius = 1.7,
 start = 4 * pi / 3, cex = 1.3, clockwise = FALSE)
```

---

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,pos=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
pos	position of the axis (see axis).
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

Details

The 'pos' argument is not needed unless the user has specified a different position from the default for the axis to be broken.

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)
```

`barlabels` *Label the bars on a barplot*

Description

Displays labels on a plot, usually a bar plot.

Usage

```
barlabels(xpos,ypos,labels=NULL,cex=1,prop=0.5,miny=0,offset=0,nobox=FALSE,...)
```

Arguments

`xpos`	A vector, matrix or data frame of x positions for the labels.
`ypos`	A vector, matrix or data frame of y values for the labels.
`labels`	The labels to display. Defaults to the values of ypos.
`cex`	Relative size of the labels. See '`text`'.
`prop`	The proportion of '`ypos`' at which to place the labels. Defaults to 0.5 (the middle).
`miny`	The minimum value at which to display labels.
`offset`	Amount to horizontally offset successive labels in case of vertical overlaps.
`nobox`	Whether to call sampboxed.labels or '`text`'.
`...`	Extra arguments passed to '`boxed.labels`' or '`text`'.

Details

'`barlabels`' places labels on a plot at horizontal positions '`xpos`' and vertical positions '`ypos`' * '`prop`'. The typical use of this function is to place labels on bars, by default in the middle of the bars.

To put labels just over the tops of the bars, set '`prop`' to 1 and add a constant amount to '`ypos`'.

Value

nil

Author(s)

Jim Lemon

See Also

boxed.labels

Examples

```
heights<-c(14,20,9,31,17)
barpos<-barplot(heights,main="A redundant bar plot")
# show the usual value labels on the bars
barlabels(barpos,heights)
# now with stacked bars and offsets
heights<-matrix(sample(c(1,2,10,15),20,TRUE),ncol=4)
barpos<-barplot(heights,main="A redundant stacked bar plot")
barlabels(barpos,heights,offset=0.1)
# do it again without stacking
barpos<-barplot(heights,main="An unstacked redundant bar plot",
 beside=TRUE)
barlabels(barpos,heights)
# finally use barp for the plot
barpos<-barp(heights,main="A fourth and final bar plot",col=2:6,
 names.arg=paste("Day",1:4))
barlabels(barpos$x,barpos$y,matrix(LETTERS[1:5],nrow=5,ncol=4))
```

---

barNest *Display a nested breakdown of numeric values*

Description

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

Usage

```
barNest(formula=NULL,data=NULL,FUN=c("mean","sd","sd","valid.n"),ylim=NULL,
main="",xlab="",ylab="",shrink=0.1,errbars=FALSE,col=NA,
labelcex=1,lineht=NULL,showall=TRUE,Nwidths=FALSE,barlabels=NULL,
showlabels=TRUE,mar=NULL,arrow.cap=NULL,trueval=TRUE)
```

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'.
FUN	The functions to apply to x.
ylim	Optional y limits for the plot, usually necessary for counts.
main	Title for the plot.
xlab,ylab	Axis labels for the plot. The x axis label is typically blank
shrink	The proportion to shrink the width of the bars at each level.
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.
Nwidths	Whether to scale the widths of the bars to the number of observations.
barlabels	Optional group labels that may be useful if the factors used to break down the numeric variable are fairly long strings.
showlabels	Whether to display the labels below the bars.
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, calculated on the basis of the number of bars in the final breakdown if NA.
trueval	If this is not NA, the call to 'brkdnNest' will return the proportions of the response variable that are equal to 'trueval'. See Details.

Details

'barNest' displays a bar plot illustrating the hierarchic breakdown of the elements of a data frame. The breakdown is performed by 'brkdnNest' and the actual display is performed by 'drawNestedBars'. The heights of the bars will be proportional to the values returned by the first function in 'FUN'. 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.

'barNest' assumes that there will be four breakdowns in the list returned by 'brkdnNest' in the order summary measure, upper dispersion value, lower dispersion value and number of valid observations. If 'Nwidths=FALSE', it may work with only three and if 'errbars=FALSE' as well, it may work with only one.

If 'Nwidths=TRUE', the bar widths will be scaled to the relative number of observations per group. When the numbers of observations are very different, the labels for those bars with small numbers of observations will probably overlap.

A number of functions can be passed in the 'FUN' argument. Three functions, 'propbrk', 'sumbrk' and 'valid.n' will work as summary measures, giving proportions or sums of particular values of a discrete variable and counts in each group and subgroup respectively. Binomial confidence limits can be added to the proportions returned by 'propbrk' with 'binciWl' and 'binciWu' as in the second last example. If 'valid.n' is the first element of 'FUN', the "overall" bar and label will be suppressed, as they are not informative. It is up to the user to decide whether any "error bars" displayed are meaningful.

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 (last 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. As a 'barNest' plot with more than a few factors and levels in each factor is quite dense, easily distinguished colors for each level of the breakdown may be preferable.

As with some other plots, trying to cram too much information into a single illustration may not work well.

Value

The summary list produced by brkdnNest.

Author(s)

Jim Lemon and Ofir Levy

References

Lemon, J. & Levy, O. (2011) barNest: Illustrating nested summary measures. Statistical Computing and Graphics Newsletter of the American Statistical Association, 21(2): 5-10.

See Also

brkdnNest, drawNestedBars, superbarplot(UsingR)

Examples

```
# recreate the Titanic data frame and show the three way breakdown
titanic<-data.frame(
 class=c(rep("1st",325),rep("2nd",285),rep("3rd",706),rep("Crew",885)),
 age=c(rep("Adult",319),rep("Child",6),rep("Adult",261),rep("Child",24),
 rep("Adult",627),rep("Child",79),rep("Adult",885)),
 sex=c(rep("M",175),rep("F",144),rep("M",5),rep("F",1),
 rep("M",168),rep("F",93),rep("M",11),rep("F",13),
 rep("M",462),rep("F",165),rep("M",48),rep("F",31),
 rep("M",862),rep("F",23)),
 survived=c(rep("Yes",57),rep("No",118),rep("Yes",140),rep("No",4),rep("Yes",6),
 rep("Yes",14),rep("No",154),rep("Yes",80),rep("No",13),rep("Yes",24),
 rep("Yes",75),rep("No",387),rep("Yes",76),rep("No",89),
 rep("Yes",13),rep("No",35),rep("Yes",14),rep("No",17),
 rep("Yes",192),rep("No",670),rep("Yes",20),rep("No",3)))
require(plotrix)
titanic.colors<-list("gray90",c("#0000ff","#7700ee","#aa00cc","#dd00aa"),
 c("#ddcc00","#ee9900"),c("pink","lightblue"))
barNest(survived~class+age+sex,titanic,col=titanic.colors,showall=TRUE,
 main="Titanic survival by class, age and sex",ylab="Proportion surviving",
 FUN=c("propbrk","binciWu","binciWl","valid.n"),shrink=0.15,trueval="Yes")
barNest(survived~class+age+sex,titanic,col=titanic.colors,showall=TRUE,
 main="Titanic survival by class, age and sex (scaled bar widths)",
 ylab="Proportion surviving",FUN=c("propbrk","binciWu","binciWl","valid.n"),
 shrink=0.15,trueval="Yes",Nwidths=TRUE)
# now show the actual numbers of passengers
barNest(survived~class+age+sex,titanic,col=titanic.colors,showall=TRUE,
 main="Titanic passengers and crew by class, age and sex",
 ylab="Number",FUN="valid.n",shrink=0.15)
# to see this properly displayed, start a wide plot window
# x11(width=10)
```

```
test.df<-data.frame(Age=rnorm(100,35,10),
 Sex=sample(c("Male","Female"),100,TRUE),
 Marital=sample(c("Div","Mar","Sing","Wid"),100,TRUE),
 Employ=sample(c("FT","PT","Un"),100,TRUE))
test.col<-list(Overall="gray",Sex=c("pink","lightblue"),
 Marital=c("mediumpurple","orange","tan","lightgreen"),
 Employ=c("#1affd8","#caeecc","#ff90d0"))
barNest(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
 main="Mean age by subgroups",errbars=TRUE,col=test.col)
barNest(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
 main="Mean age by subgroups (widths scaled to Ns)",errbars=TRUE,col=test.col,
 Nwidths=TRUE)
# set up functions for 20th and 80th percentiles
q20<-function(x,na.rm=TRUE) return(quantile(x,probs=0.2,na.rm=TRUE))
q80<-function(x,na.rm=TRUE) return(quantile(x,probs=0.8,na.rm=TRUE))
# show the asymmetric dispersion measures
barNest(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Mean age (years)",
 main="Use median and quantiles for dispersion",
 FUN=c("median","q80","q20","valid.n"),
 errbars=TRUE,col=test.col)
barNest(formula=Employ~Sex+Marital,data=test.df,ylab="Proportion unemployed",
 main="Proportion unemployed by sex and marital status",
 FUN=c("propbrk","binciWu","binciWl","valid.n"),
 errbars=TRUE,col=test.col,trueval="Un")
barNest(formula=Employ~Sex+Marital,data=test.df,ylab="Proportion unemployed",
 main="Proportion unemployed by sex and marital status (scaled bar widths)",
 FUN=c("propbrk","binciWu","binciWl","valid.n"),
 errbars=TRUE,col=test.col,trueval="Un",Nwidths=TRUE)
barNest(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Counts",
 main="Show the counts in subgroups (final level only)",FUN="valid.n",
 col=test.col,showall=FALSE,ylim=c(0,10))
barNest(formula=Age~Sex+Marital+Employ,data=test.df,ylab="Counts",
 main="Show all the counts in subgroups",FUN="valid.n",mar=c(5,5,4,2),
 col=test.col)
```

---

barp *A bar plotting routine*

Description

Display a bar plot

Usage

```
barp(height,width=0.4,names.arg=NULL,legend.lab=NULL,legend.pos=NULL,
col=NULL,border=par("fg"),main=NULL,xlab="",ylab="",xlim=NULL,ylim=NULL,
x=NULL,staxx=FALSE,staxy=FALSE, height.at=NULL,height.lab=NULL,
cex.axis=par("cex.axis"),pch=NULL,cylindrical=FALSE,shadow=FALSE,
do.first=NULL,ylog=FALSE,srt=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. 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.
x	Optional horizontal positions for the bars. Defaults to 1:length(height).
staxx,staxy	Whether to use staxlab to stagger the X or Y axis tick labels. Can also omit the X or Y axes.
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.
pch	Symbol(s) to fill the bars. See Details.
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 string that will be evaluated before anything else is displayed on the plot. Useful for background colors or lines.
ylog	Logical for whether a log scale is to be used. see details.
srt	Rotation of axis labels if staxx or staxy is TRUE (see 'staxlab').
...	arguments passed to 'plot'.

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 by default, and so both the width and spacing of the bars are controlled by a single number. If the user passes explicit 'x' values, those values will determine the spacing. If 'height' is a vector, single bars representing each value will be displayed centered at '1:length(height)' unless the user has specified 'x' values. If 'height' is a matrix, 2D array, 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. Explicit x values cannot be used with a matrix, however, by adjusting the values of x, grouped bars can be displayed.

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', as can a 2D array returned from the 'by' function.

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. If 'srt' is not NULL, these labels will be rotated counterclockwise by that value as angles in degrees instead of staggered.

If 'staxx' or 'staxy' are set to NA, the respective axis will not be displayed.

'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 'dispersion', but many advise against this.

If 'legend.lab' is not NULL, a legend will be displayed. If 'legend.pos' is NULL, '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.

The 'ylog' argument produces a log scale on the y axis. Currently, neither 'pretty' nor 'axTicks' seems to produce a nice set of axis ticks, so it is best to pass the positions of these in 'height.at'.

If the 'pch' argument is not NULL, barp will display white bars filled with the symbols specified in 'pch'. With grouped bars, this must be a matrix with the same form as the 'col' argument. This option allows a black and white bar plot to be produced.

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=list(x=2,y=4.5),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!)
sexratio<-c(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")
# xrange is used to get the colors to match the 0-100% scale
barp(funrating,names.arg=funstudy,main="Fun ratings for various areas of study",
 col=color.scale(sexratio,c(0.2,1),c(0.2,0.4),c(1,0.4),xrange=c(0,1)),
 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 with a shaded background
# notice how the expression uses local variables inside the barp function
gradbg<-"gradient.rect(xlim[1],ylim[1],xlim[2],ylim[2],
 c(1,0.5,1),c(1,0.5,1),c(1,0.5,1),gradient=\"y\",nslices=100)"
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,do.first=gradbg,
 main="Multiple histogram using barp",xlab="Bins",ylab="Frequency")
legend(3.8,50,c("h1","h2","h3"),fill=2:4)
# now display a positive/negative plot
barp(c(2,-3,4,-5,6,-7,8),main="Positive/negative plot",
 xlab="Alternating colors",ylab="For alternating values",
 col=2+(c(2,-3,4,-5,6,-7,8)>0))
```

`battleship.plot` *Display a matrix of values as the widths of stacked rectangles*

Description

'`battleship.plot`' displays a matrix of rectangles, with widths proportional to the values in '`x`'. The values are scaled so that half the width of the largest rectangle is equal to '`maxxspan`' in user units. This prevents the rectangles from overlapping. The user can adjust the spacing of the stacks of

rectangles by changing 'maxxspan'. Similarly, maxyspan controls the spacing between rectangles in the vertical direction.

The labels for each stack of plots (the columns of x) are displayed at the top of the plot, angled at 45 degrees. The labels for each row of rectangles in the stacks (the rows of x) are displayed at the left. Long labels for either may require adjusting the 'mar' argument.

The function will try to extract the labels 'xaxlab' and 'yaxlab' from the matrix column and row names respectively if none are passed.

Usage

```
battleship.plot(x,mar=c(2,5,5,1),col="white",border="black",
main="",xlab="",ylab="",xaxlab=NULL,yaxlab=NULL,cex.labels=1,
maxxspan=0.45,maxyspan=0.45)
```

Arguments

x	A matrix or data frame containing numeric values. See the example.
mar	Margins for the plot.
col	The fill colors for the rectangles.
border	The border colors for the rectangles.
main	The title for the plot (i.e. 'main').
xlab,ylab	The x and y axis labels.
xaxlab,yaxlab	Optional labels for the rows and columns.
cex.labels	Character expansion for the row and column labels.
maxxspan,maxyspan	Scaling factor for the widths and heights of the rectangles so that they don't overlap.

Value

nil

Author(s)

Jim Lemon - thanks to Adam Maltese for the suggestion

See Also

plot, staxlab

Examples

```
x<-matrix(sample(10:50,100,TRUE),10)
xaxlab=c("One","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten")
yaxlab=c("First","Second","Third","Fourth","Fifth","Sixth","Seventh",
 "Eighth","Ninth","Tenth")
battleship.plot(x,xlab="The battle has just begun",main="Battleship1",
 xaxlab=xaxlab,yaxlab=yaxlab)
```

`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)
```

`binciW` *Binomial confidence limits*

Description

Calculates binomial confidence limits using the Wilson approximation.

Usage

```
binciW(x,n,alpha=0.05,cc=FALSE)
```

Arguments

`x`	The number of successes or failures for which the CI is to be calculated.
`n`	The number of trials as above.
`alpha`	The desired coverage - 0.05 produces 95 percent coverage
`cc`	Whether to apply a continuity correction

Details

'`binciW`' calculates binomial confidence limits for the given number of successes and trials. It is mainly to allow binomial confidence limits to be calculated in the '`brkdnNest`' function, which is why the upper and lower CIs are called separately.

Value

The lower and upper binomial confidence limits

Author(s)

Jim Lemon

See Also

binciWl, binciWu

Examples

```
binciW(5,42)
```

`binciWl` *Lower binomial confidence limit*

Description

Returns the lower binomial confidence limit using the Wilson approximation.

Usage

```
binciWl(x,n,alpha=0.05,trueval=TRUE,na.rm=TRUE)
```

Arguments

`x`	The number of successes or failures for which the CI is to be calculated.
`n`	The number of trials as above.
`alpha`	The desired coverage - 0.05 produces 95 percent coverage
`trueval`	The value representing the outcome of interest for the CI.
`na.rm`	Argument needed to make this work

Details

'`binciWl`' now calls '`binciW`' and returns the lower limit.

Value

The lower binomial confidence limit

Author(s)

Jim Lemon

See Also

binciWu

Examples

```
binciWl(c(rep(5,TRUE),rep(37,FALSE)))
```

`binciWu` *Upper binomial confidence limit*

Description

Returns the upper binomial confidence limit using the Wilson approximation.

Usage

```
binciWu(x,n,alpha=0.05,trueval=TRUE,na.rm=TRUE)
```

Arguments

`x`	The number of successes or failures for which the CI is to be calculated.
`n`	The number of trials as above.
`alpha`	The desired coverage - 0.05 produces 95 percent coverage
`trueval`	The value representing the outcome of interest for the CI.
`na.rm`	Argument needed to make this work

Details

'`binciWu`' now calls '`binciW`' and returns the upper limit.

Value

The upper binomial confidence interval

Author(s)

Jim Lemon

See Also

binciWl

Examples

```
binciWl(c(rep(5,TRUE),rep(37,FALSE)))
```

box.heresy *Display a sort of box plot*

Description

'box.heresy' displays a box plot in which a symbol represents a measure of central tendency, a surrounding box that represents an "inner" measure of dispersion (e.g. standard error) and whiskers represent an "outer" measure of dispersion (e.g. standard deviation). The function is pretty basic at this time and will probably change a bit.

The argument "intervals" is particularly important, and can wreak havoc on the resulting plot. The default of FALSE means that the values passed to the inner and outer measures of dispersion are absolute, not intervals away from the measure of central tendency. Mixing absolute and relative values will always lead to errors and typically a very strange looking plot. It is probably easiest to calculate the absolute values before calling box.heresy. The first and second examples show how intervals=FALSE and intervals=TRUE can be used.

One of the first changes is to allow varying box widths. The user can specify the box widths as a vector of numeric values at least as long as the number of boxes to be displayed. The usual reason for doing this is to display widths that are proportional to the number of observations. A useful start is to pass 'boxwidth' as the number of observations and let the function work it out.

Usage

```
box.heresy(x,y,uinner,linner,ulim,llim,boxwidth=NULL,
 intervals=FALSE,arrow.cap=NULL,pch=22,main="",xlab="",ylab="",
 xaxlab=NULL,col="white",do.first=NULL,...)
```

Arguments

x,y	Vectors of numeric values representing measures of central tendency.
uinner,linner	Vectors of numeric values representing "inner" measures of dispersion.
ulim,llim	Vectors of numeric values representing "outer" measures of dispersion.
boxwidth	Optional widths for the boxes.
intervals	Whether the values for dispersion are intervals (TRUE) or absolute limits (FALSE).
arrow.cap	The width of the cap on the "whiskers" relative to the width of the plot. Defaults to the same width as the outer box.
pch	The symbol to be used to represent the measure(s) of central tendency in the box.
main	The title for the plot (i.e. 'main').
xlab,ylab	The x and y axis labels.
xaxlab	Optional labels for the boxes.
col	The fill colors for the "inner" rectangles.
do.first	An expression that will be evaluated before anything is displayed.
...	additional arguments passed to the 'dispersion' function.

Value

nil

Author(s)

Jim Lemon - thanks to Gianni Lavaredo for the suggestion

See Also

plot, boxplot

Examples

```
y1<-runif(20,2,10)
y2<-rnorm(30,6,2)
y3<-sample(0:20,40,TRUE)
Ns<-c(20,30,40)
ymean<-c(mean(y1),mean(y2),mean(y3))
y1inner<-quantile(y1,probs=c(.16,.84))
y2inner<-c(ymean[2]+sd(y2),ymean[2]-sd(y2))
y3inner<-quantile(y3,probs=c(.16,.84))
uinner<-c(y1inner[1],y2inner[1],y3inner[1])
linner<-c(y1inner[2],y2inner[2],y3inner[2])
ulim<-c(max(y1),max(y2),max(y3))
llim<-c(min(y1),min(y2),min(y3))
box.heresy(ymean,uinner=uinner,linner=linner,ulim=ulim,llim=llim,boxwidth=Ns,
 main="Boxplot of means, central spread and range",xlab="Distribution",
 xaxlab=c("Uniform","Normal","Sample"))
y1outer<-
y<-runif(5)
ulim<-runif(5)
llim<-runif(5)
uinner<-ulim/2
linner<-llim/2
box.heresy(y,uinner=uinner,linner=linner,ulim=ulim,llim=llim,
 intervals=TRUE,main="The heretical boxplot",
 xlab="Number of observations",ylab="Value")
```

---

boxed.labels *Place labels in boxes*

Description

Places labels in boxes on an existing plot

Usage

```
boxed.labels(x,y=NULL,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,xlog=FALSE,ylog=FALSE,...)
```

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.
xlog	Whether the X axis is a log axis
ylog	Whether the Y axis is a log axis
...	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. With the default 'textcol=NA', the function tries to work out whether white or black text will be more easily read based on the background color and displays the text accordingly. If the user specifies text colors in the additional arguments, these colors will override the automatic white/black above - see the last example.

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 - thanks to Thorn Thaler for the code allowing user-specified text colors and Flemming Skjoth for the log axis correction

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<-barp(c(1,3,2,4))
boxed.labels(xpos$x,0.5,nums[1:4])
# and add labels below the x axis ticks
boxed.labels(xpos$x,-0.4,c("First","Second","Third","Fourth"))
# 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="yellow")
```

---

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

Arguments

vars	The names or indices of one or more columns in a data frame. The columns must contain numeric data. If only one variable is to be broken down, vars can be a formula.
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.
yat	Optional y axis tick positions.
...	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.

If only one numeric variable is to be broken down, 'vars' may be a formula like 'var~groups+obs'. The position of the two factors to break down the variable is fixed - the second term will be interpreted as "groups" and the third, if present, will be interpreted as "obs".

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

Author(s)

Jim Lemon

See Also

dispersion

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,data=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)
```

---

brkdnNest *Perform a nested breakdown of numeric values*

Description

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

Usage

```
brkdnNest(formula,data,FUN=c("mean","sd","sd","valid.n"),label1="Overall",
 trueval=TRUE)
```

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'.
FUN	The functions to be applied to successive breakdowns.
label1	The label to use for the overall value of the first function.
trueval	The value to use in calculating proportions or sums of a categorical response variable. See Details.

Details

'brkdnNest' performs a nested breakdown of an element of a data frame by one or more categorical elements. For each category and optional subcategories, the variable on the left of the formula is summarized as specified by the functions named in 'FUN'.

If 'trueval' is not NA, brkdnNest will calculate the proportion of 'trueval' values in the response variable out of the total valid responses. If the function 'valid.n' is the first function in 'FUN', the counts of the groups and subgroups will be returned.

Two specialized summary functions are defined within 'brkdnNest'. 'sumbrk' returns the count of values in a factor equal to 'trueval', and 'propbrk' returns the proportion of values equal to 'trueval'. Be aware that if a categorical variable is specified on the left of the formula, functions which expect numeric data such as 'mean' should not be included in 'FUN'.

The user should take care when specifying different summary functions. 'barNest' expects a summary measure as the first component of the list and measures of dispersion as the second and third. If two different measures of dispersion are passed, the first must calculate the upper and the second the lower measure.

Value

A list with as many elements as there are functions in 'FUN'. It is probably best to always specify four functions (summary measure, upper dispersion measure, lower dispersion measure and number of valid observations) even if this is redundant as in the default.

This function is similar to 'brkdn' in the **prettyR** package, but is structured to be used with the 'barNest' function. It produces one or more measures for the overall data, then the subsets of the data defined by the first variable to the right of the tilde, then the subsets defined by the first and second variable, and so on.

Author(s)

Jim Lemon

See Also

by

Examples

```
brkdntest<-data.frame(Age=rnorm(100,25,10),
 Sex=factor(sample(c("M","F"),100,TRUE)),
 Marital=sample(c("M","X","S","W"),100,TRUE),
 Employ=sample(c("FT","PT","NO"),100,TRUE))
brkdnNest(formula=Age~Sex+Marital+Employ,data=brkdntest)
# show the proportion of unemployed with binomial confidence intervals
brkdnNest(formula=Employ~Sex+Marital,data=brkdntest,
 FUN=c("propbrk","binciWu","binciWl"),trueval="NO")
```

bumpchart *Display a "bumps" (sequential ranking) 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=rep(rownames(y),2),rank=TRUE,
 mar=c(2,8,5,8),pch=19,col=par("fg"),lty=1,lwd=1,arrows=FALSE,...)
```

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. Alter to your taste.
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.
arrows	Whether to join the points with lines (FALSE) or arrows (TRUE).
...	Additional arguments passed to 'matplot' or 'arrows'.

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. These labels may now be different for each side of the plot, for example if the values of 'y' are to be included. Remember that due to the transposition of the values for plotting, the labels on the right have to precede those on the left - see the second example.

This type of plot is often used to show the changing positions of entities over time, like the ranking in surveys in different years. For a similar, but more flexible plot, see ladderplot.

Because of the way 'matplot' plots the values, the order of everything is reversed. As the typical display of ranks is with rank 1 at the top, the actual rank positions are used to plot the values. This places the lowest scores at the bottom of the plot and the highest at the top.

Any arguments that are included in '...' will be passed to 'matplot' if the 'arrows' argument is FALSE, and to the 'arrows' function if the 'arrows' argument is TRUE as in the first example.

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",
 arrows=TRUE,length=0.2)
vallab<-c(paste(educattn[,2],rownames(educattn),sep="-"),
 paste(rownames(educattn),educattn[,1],sep="-"))
# now show the raw percentages and add central ticks
bumpchart(educattn,rank=FALSE,labels=vallab,
 main="Percentage high school completion by over 25s",
 lty=1:10,lwd=1,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)
```

---

categoryReshape *Convert object label/attribute label coding.*

Description

Convert object label/attribute label coding to an object by attribute data frame.

Usage

```
categoryReshape(x)
```

Arguments

x	A matrix or data frame with at least two columns.

Details

'categoryReshape' attempts to convert the first two columns of its input into a data frame in which rows represent objects and columns attributes. For each object, a value of 1 indicates that the object has that attribute, and a value of 0 that it does not. In set membership terms, a 1 indicates that the object is a member of that set and a 0 that it is not.

Value

A data frame (see Details).

Author(s)

Jim Lemon

See Also

makeIntersectList

Examples

```
ns<-sample(1:8,20,TRUE)
objects<-0
for(i in 1:length(ns)) objects<-c(objects,rep(i,ns[i]))
attributes<-"Z"
for(i in 1:length(ns)) attributes<-c(attributes,sample(LETTERS[1:8],ns[i]))
setdf<-data.frame(objects[-1],attributes[-1])
categoryReshape(setdf)
```

---

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,pch=21,vgrid=NA,hgrid=NA,gridcol="lightgray",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.
pch	The symbols to use when plotting midpoints.
vgrid	Optional vertical line(s) to display on the plot. Defaults to NA (none).
hgrid	Optional horizontal grid lines to display on the plot. Defaults to NA (none).
gridcol	The color for the vgrid and hgrid lines.
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.

The 'vgrid' argument is usually used to display an average value for all of the midpoints. If one or more values are passed in this argument, they will be displayed as vertical lines spanning the plot. The 'hgrid' argument acts like the 'grid' function, drawing dashed horizontal lines across the plot. If 'hgrid=NULL', these lines will be drawn under the values displayed, which will be 1 to the number of values on the vertical axis. The user can pass explicit values if desired. With horizontal and optionally vertical grid lines, the centipede plot is practically equivalent to a dotplot with error bars.

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 there are no values for number of valid observations, just pass vector of blank strings with the 'right.labels' argument. 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)
# now leave out the number of valid observations, pass x labels and no right labels
centipede.plot(segs[1:3,],main="Test centipede plot",vgrid=0,mar=c(4,5,3,2),
 left.labels=paste("X",1:40,sep=""),right.labels=rep("",40))
```

---

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 '\dots'
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.

If 'add=TRUE' is passed as one of the additional arguments, the values will be added to the current plot. If a 'radial.lim' argument was passed on the initial plot, it must be passed again to add values or the values will be displayed incorrectly.

Usage

```
clock24.plot(lengths,clock.pos,labels=0:23,minutes=FALSE,
 hm2dec=FALSE,label.pos=NULL,rp.type="r",loglen=FALSE,explab=FALSE,...)
```

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.
minutes	Whether to add minutes (".00") to the labels.
hm2dec	Whether to convert HH:MM clock positions to decimal hours.
label.pos	Radial positions of the labels.
rp.type	Whether to plot radial lines, symbols or a polygon.
loglen	Whether to log transform the 'length' values. Only base 10 logs are available.
explab	Whether to use the default fixed (FALSE) or exponential (TRUE) notation for the radial labels.
...	additional arguments are passed to 'radial.plot' and then to 'plot'.

Value

A list of the parameters altered by radial.plot.

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
oldpar<-clock24.plot(testlen[7:19],testpos[7:19],
 main="Test Clock24 daytime (symbols)",
 point.col="blue",rp.type="s",lwd=3)
# reset everything
par(oldpar)
```

---

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. Must have both values.
`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 - thanks to Markus Elze for the test for a current graphics device

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),
 away=rep(0.2,2))
plot(clusteredpoints,col=clusteredpoints$col,
 main="Cluster overplot test")
```

---

clustered.dotplots *Display the frequencies of two categories*

Description

'clustered.dotplots' displays a contingency table as clusters of symbols on a plot. It expects 'xgroup' and 'ygroup' to contain all or some of the combinations of their unique values. It also expects 'freq' to contain the number of instances of each combination.

Usage

```
clustered.dotplots(xgroup, ygroup, freq, type = "circles",
main="",xlab="",ylab="",x_las=1,y_las=1,axes=TRUE,size=1,...)
```

Arguments

xgroup,ygroup	Vectors that specify the two groupings to be displayed (see Details).
freq	The frequencies in the two groupings.
type	The type of symbols to use as "dots".
main,xlab,ylab	As in plot.
x_las,y_las	Orientation of the axis tick labels.
axes	Whether to display axes.
size	Spacing for the clusters.
...	additional arguments passed to "points".

Value

nil

Author(s)

Darshan Baral

See Also

cluster.overplot

Examples

```
df <- structure(list(set = c("09t0101 TJ", "09t0102 MW", "09t0201 EH",
"09t0202 NH", "09t0101 TJ", "09t0102 MW", "09t0201 EH", "09t0202 NH",
"09t0101 TJ", "09t0102 MW", "09t0201 EH", "09t0202 NH", "09t0101 TJ",
"09t0102 MW", "09t0201 EH", "09t0202 NH", "09t0202 NH"), grade = c("1",
"1", "1", "1", "2", "2", "2", "2", "3", "3", "3", "3", "4", "4",
"4", "4", "5"), freq = c(7, 8, 2, 3, 11, 4, 11, 3, 3, 8, 3, 8,
3, 9, 3, 2, 5)), .Names = c("set", "grade", "freq"), row.names = c(NA,
17L), class = "data.frame")
clustered.dotplots(xgroup = df$set, ygroup = df$grade, freq = df$freq)
clustered.dotplots(xgroup = df$set, ygroup = df$grade, freq = df$freq,
 col = "gray")
clustered.dotplots(xgroup = df$set, ygroup = df$grade, freq = df$freq,
 type = "points")
clustered.dotplots(xgroup = df$set, ygroup = df$grade, freq = df$freq,
 type = "points", pch = 19, col = "red")
# this will cause an error
# clustered.dotplots(xgroup = mtcars$cyl, ygroup = mtcars$gear,
#  freq = mtcars$carb)
# how to fix it
cumcars<-by(mtcars$carb,list(mtcars$cyl,mtcars$gear),valid.n)
mtcars2<-data.frame(cyl=NA,gear=NA,carb=NA)
rownum<-1
for(cyl in dimnames(cumcars)[[1]]) {
 for(gear in dimnames(cumcars)[[2]]) {
  if(!is.na(cumcars[cyl,gear])) {
   mtcars2[rownum,]<-c(as.numeric(cyl),as.numeric(gear),cumcars[cyl,gear])
   rownum<-rownum+1
  }
 }
}
clustered.dotplots(xgroup = mtcars2$cyl, ygroup = mtcars2$gear,
 freq = mtcars2$carb,main="Cars by number of cylinders and gears",
 xlab="Number of cylinders",ylab="Number of gears",type="points",pch=5)
```

`color.axis` *Display an axis in a specified color*

Description

'`color.axis`' displays an axis in the specified color.

Usage

```
color.axis(side=1,at=NULL,labels=TRUE,axlab=NA,axlab.at=NA,
 col=par("fg"),cex.axis=par("cex.axis"),cex=par("cex"))
```

Arguments

side	Which axis - see axis.
at	Positions for the tick labels.
labels	Tick labels.
axlab	Optional axis label.
axlab.at	Where to position the axis label - defaults to centered.
col	Color for the axis.
cex.axis	Character expansion for the tick labels.
cex	Character expansion for the axis label.

Value

nil

Author(s)

Jim Lemon

---

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 specified 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 I have only found one use for this. If the user wants the labels at the intersection of the boxes rather than in the center, see the alternative specification for the labels in the example (thanks Claudia Tebaldi). To have complete control over the labels, see gradient.rect and text or mtext.

'`colorlegend`' in the **shape** package offers a different approach, creating a large number of colors from a color generating function (a bit like '`color.gradient`') and then allowing the user to specify tick marks at arbitrary points along the color bar.

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")
# this will put the labels at the intersections
# col.labels<-c("","Cold","","Warm","","Warmer","","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 using RGB, HSV or HCL

Usage

```
color.scale(x,cs1=c(0,1),cs2=c(0,1),cs3=c(0,1),alpha=1,
 extremes=NA,na.color=NA,xrange=NULL,color.spec="rgb")
```

Arguments

x	a numeric vector, matrix or data frame
cs1,cs2,cs3	color parameters for scaling 'x'
alpha	Value for transparency in colors. If more than one value is passed, the alpha values will be transformed like the colors.
extremes	The colors for the extreme values of 'x' (RGB only).
na.color	The color to use for NA values of 'x'.

xrange	An explicit range to use in the transformation.
color.spec	The color specification to use in the transformation. Anything other than "rgb", "hsv" or "hcl" will almost certainly fail.

Details

'color.scale' calculates a sequence of colors by a linear transformation of the numeric values supplied into the ranges for the three color parameters. 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 for the RGB or HSV specifications, and between 0 and 360 (cs1) and 0 to 100 (cs2 and cs3) for the HCL specifications.

IMPORTANT: If 'x' has fewer values than the number of values in the color parameters, it will usually return incorrect colors. This is usually only a problem when using 'color.legend' with a small number of rectangles in the legend as 'color.legend' calls 'color.scale' to calculate the color rectangles.

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. Note that this forces the color specification to RGB.

If the user wants to specify a range of values with 'xrange', it must at least include the range of x values. This can be useful when there is a notional range like 0-100% that the values do not cover, or when several series of values with different ranges are to be assigned the same color scale.

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. I may get around to adding an argument to do this in one shot. Until then, see the second example for 'color2D.matplot' and also the 'diverge.hcl' and 'diverge.hsv' functions in the **colorspace** package.

When passing more than one alpha value, it will be transformed like the colors. This allows matrices with concentrations of high values to be overplotted to illustrate group locations and separations. See the iris example in '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.

There are quite a few R functions that transform numeric values into colors or produce colors that can be used to represent values. Two packages that might be of interest are **RColorBrewer** and **colourschemes**. See the last example for approximating other color scales with 'color.scale'.

Author(s)

Jim Lemon

See Also

rescale, col2rgb, smoothColors

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)
plot(1:10,rep(1:3,length.out=10),axes=FALSE,type="n",xlim=c(0,11),ylim=c(0,4),
 main="Test of RGB, HSV and HCL",xlab="",ylab="Color specification")
axis(2,at=1:3,labels=c("HCL","HSV","RGB"))
points(1:10,rep(1,10),pch=19,cex=8,col=color.scale(1:10,c(0,300),35,85,
 color.spec="hcl"))
points(1:10,rep(2,10),pch=19,cex=8,col=color.scale(1:10,c(0,1),
 0.8,1,color.spec="hsv"))
points(1:10,rep(3,10),pch=19,cex=8,col=color.scale(1:10,c(1,0.5,0),
 c(0,0.5,0),c(0,0,1),color.spec="rgb"))
## Not run:
 # requires viridisLite
 library(viridisLite)
 plot(0,xlim=c(-1,1),ylim=c(-1,1),type="n",axes=FALSE,
  main="Approximating other color scales",xlab="",ylab="")
 gradient.rect(-1,0.8,1,0.95,nslices=50,
  col=color.scale(1:50,1,
  c(0,0.3,0.6,0.8,1,1),
  c(0,0,0,0,0,1)))
 text(0,1,"color.scale")
 gradient.rect(-1,0.65,1,0.8,col=heat.colors(50))
 text(0,0.6,"heat.colors")
 gradient.rect(-1,0.3,1,0.45,nslices=50,
  col=color.scale(1:50,c(0,0.2,0.9,0.95,0.95),
  c(0.7,0.8,0.9,0.7,0.95),
  c(0.1,0,0,0.35,0.95)))
 text(0,0.5,"color.scale")
 gradient.rect(-1,0.15,1,0.3,col=terrain.colors(50))
 text(0,0.1,"terrain.colors")
 gradient.rect(-1,-0.2,1,-0.05,nslices=50,
  col=color.scale(1:50,c(0.3,0,0.3,0.1,1,0.95,1),
  c(0,0.3,0.9,1,1,0.85,0.85),
  c(1,1,0.9,0.1,0,0.5,0.5)))
 text(0,0,"color.scale")
 gradient.rect(-1,-0.35,1,-0.2,col=topo.colors(50))
 text(0,-0.4,"topo.colors")
 gradient.rect(-1,-0.7,1,-0.55,nslices=50,
  col=color.scale(1:50,c(0.3,0.2,0,0.4,0.95),
  c(0.1,0.3,0.6,0.75,0.95),
  c(0.3,0.6,0.5,0.4,0)))
 text(0,-0.5,"color.scale")
 gradient.rect(-1,-0.85,1,-0.7,col=viridis(50))
```

```
  text(0,-0.9,"viridis")

## End(Not run)
```

`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)
# now color the lines to show whether each step went away from
# or toward the starting position
color.scale.lines(x,y,col=2+(diff(xydist)>0))
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,cs1=c(0,1),cs2=c(0,1),cs3=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=NA,vcex=1,
 border="black",na.color=NA,xrange=NULL,color.spec="rgb",yrev=TRUE,
 xat=NULL,yat=NULL,Hinton=FALSE,add=FALSE,...)
```

Arguments

x	data values
cs1,cs2,cs3	the color parameters that will be scaled to represent the range of numeric values. (see 'color.scale')
extremes	The colors for the extreme values of 'x'. Takes precedence over the color ranges.
cellcolors	A precalculated matrix of cell colors. This must have the same number of rows and columns as the matrix or it will be uninformative. It can be a vector, but be careTakes 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. This will force the color specification to "rgb", so if this is different from the color specification requested, 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. If NA, the values are displayed in black or white depending upon the darkness of the cell color.
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'.
xrange	An explicit range for the transformation of colors. see 'color.scale'
color.spec	The color specification system to use.
yrev	Whether to reverse the order of the y-axis to display the cells in "reading" order (left to right and top to bottom) if TRUE, or in the order of a typical plot (left to right and bottom to top) if FALSE.
xat,yat	Values at which to place tick marks to override 'pretty'.
Hinton	Whether to display a Hinton diagram in which the magnitude of cell values is proportional to the size of the squares and the sign is indicated by the color of the squares.
add	If TRUE, no plot is created and the rectangles are displayed over whatever is on the current device (see the "iris" example).
...	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. To get the rows beginning at the bottom, use 'yrev=FALSE'. 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' and 'show.legend' are also used to control the number of decimal places displayed if the values or legend are shown. 'TRUE' will give one decimal place, '2' two, and so on.

if 'Hinton' is TRUE, a Hinton diagram in which the sizes of the squares are proportional to the absolute value of 'x' and the colors of the squares indicate the sign of the 'x' values will be displayed. This only works with squares.

If 'add' is true, the color matrix is added to the current plot. This is probably only useful when displaying plots that are mostly transparent.

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, fill.corner, image

Examples

```
x<-matrix(rnorm(1024),nrow=32)
# simulate a correlation matrix with values -0.5 to 0.5
x<-rescale(x,c(-0.5,0.5))
# add a column with the extreme values (-1,1) to calculate
# the colors, then drop the extra column in the result
cellcol<-color.scale(cbind(x,c(-1,rep(1,31))),c(0,1),0,c(1,0))[,1:32]
color2D.matplot(x,cellcolors=cellcol,main="Blue to red correlations")
# do the legend call separately to get the full range
color.legend(0,-4,10,-3,legend=c(-1,-0.5,0,0.5,1),
 rect.col=color.scale(c(-1,-0.5,0,0.5,1),c(0,1),0,c(1,0)),align="rb")
x<-matrix(rnorm(100),nrow=10)
# generate colors that show negative values in red to brown
# and positive in blue-green to green
cellcol<-matrix(rep("#000000",100),nrow=10)
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,-1.8,3,-1.4,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")
# Hinton diagram
border.col<-color.scale(x,extremes=2:3)
color2D.matplot(x,extremes=c(2,3),main="Hinton diagram (green +, red -)",
 Hinton=TRUE,border=border.col)
# waffle plot of percentages with two contributing elements
waffle.col<-fill.corner(c(rep("red",18),rep("blue",45)),10,10)
color2D.matplot(matrix(1:100,nrow=10),cellcolors=waffle.col,yrev=FALSE,
 border="lightgray",xlab="",ylab="",main="Waffle plot",axes=FALSE)
# coarse density plot of the iris petal data
spnames<-unique(iris$Species)
spcols<-c("red","green","blue")
matmax<-list()
cindx<-1
for(isp in spnames) {
 petal_mat<-makeDensityMatrix(iris[iris$Species == isp,"Petal.Length"],
  iris[iris$Species == isp,"Petal.Width"],
  nx=20,ny=20,xlim=c(1,7),ylim=c(0,2.5),geocoord=FALSE)
 # center the maximum markers in the cells
 matmax[[cindx]]<-lapply(find_max_cell(petal_mat),"-",0.5)
 if(isp == "setosa")
  color2D.matplot(petal_mat,main="Iris petal length by petal width",
   xlab="Petal length (cm)",ylab="Petal width (cm)",axes=FALSE,
   cellcolors=color.scale(petal_mat,extremes=spcols[cindx],alpha=c(0,1)),
   border=NA,yrev=FALSE)
 else
  color2D.matplot(petal_mat,border=NA,yrev=FALSE,add=TRUE,
  cellcolors=color.scale(petal_mat,extremes=spcols[cindx],alpha=c(0,1)))
 cindx<-cindx+1
}
axis(1,at=seq(0,20,by=3.33),labels=1:7)
axis(2,at=seq(0,20,length.out=4),labels=seq(1,2.5,by=0.5))
legend(1,6,paste0(spnames,"(",1:3,")"),fill=c("red","green","blue"))
for(cindx in 1:3)
 text(matmax[[cindx]],as.character(cindx),col="white",cex=1.5)
```

`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, becomes progressively lighter to a "highlight" at two thirds of the width or height and then darkens 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")
```

---

death_reg *Death registrations from 1996 to 2010*

Description

Death registrations for underlying cause of death as ICD-10 chapters for 1996 to 2010.

Usage

```
data(death_reg)
```

dendroPlot *Display distributions as dendrites*

Description

Display the distributions of one or more sets of points as branching (dendritic) clusters.

Usage

```
dendroPlot(x,breaks=list(10,10,10),pch=1,col=par("fg"),cex=1,nudge=NA,
 setlabels=NA,...)
```

Arguments

x	A list or data frame of numeric or factor or character columns.
breaks	A list of cutpoints to transform numeric values into factors (see cut). Must be at least one number >= 2.
pch	Symbol(s) to use in plotting the values.
col	Color(s) for the symbols.
cex	Size of the symbol(s) to use in plotting.
nudge	The amount to set each consecutive value in a category away from the center of the dendrite.
setlabels	Labels to place along the abcissa to identify the sets.
...	Other arguments passed to plot.

Details

'dendroPlot' displays the distributions of categorical values as stacks of "branches". The lengths of the branches show the number of values in each category, rather like the opposed bars in a pyramid plot, except that there is no separation of groups. The distribution of numeric values can also be displayed by passing a set of breakpoints to categorize the values. The breakpoints will usually be equidistant, but unevenly spaced breakpoints can be passed. If an element of 'x' is numeric, the values of the corresponding 'x' element will be used to place the points, but the branches will be defined as the categories formed by applying the breaks to those numeric values.

Note that in the first example, the breakpoints for the first and third elements are used to define the ten branches for each. The second element of 'x' is already categorical, so the breakpoints are ignored. When comparing distributions with very different ranges it may be necessary to adjust the breakpoints to get a satisfactory result.

Each successive point in a category is 'nudge'd away from the center of the dendrite. If 'nudge' has more than one value, the points will be nudged up and down for categorical variables to enable closer packing. The second value of 'nudge' is ignored for numeric variables. The aspect ratio of the plot, the character expansion and the nudging will have to be adjusted to give the best point spacing for most dendroPlots.

Value

nil

Note

The 'ehplot' function is a much more versatile instantiation of this type of plot. 'dendroPlot' has been retained as there are currently a few differences that some users may find valuable. However, it is not impossible that 'dendroPlot' will disappear in the future. Another very useful version of this type of plot is 'beeswarm' in the **beeswarm** package.

Author(s)

Jim Lemon

See Also

ehplot

Examples

```
x<-list(runif(90,1,3),factor(sample(LETTERS[1:10],100,TRUE)),rnorm(80,mean=5))
dendroPlot(x,xlab="Groups",ylab="Value of x",main="Test dendroPlot I")
# now apply a nudge factor and different breaks
dendroPlot(x,breaks=list(8,10,10),nudge=c(0.05,0.1),
 xlab="Groups",ylab="Value of x",main="Test dendroPlot II")
```

---

densityGrid *Display a matrix of cell values as symbols.*

Description

Displays a matrix of values as symbols on an existing image.

Usage

```
densityGrid(x,z=NULL,xrange=NA,zrange=NA,range.cex=c(1,10),
xlim=c(-180,180),ylim=c(-90,90),red=c(0,1),green=c(0,1),blue=c(0,1),alpha=1,
pch=".",geocoord=TRUE)
```

Arguments

x	Matrix of values representing counts in cells (usually locations).
z	Optional matrix of values attached to the cells in x.
xrange,zrange	Explicit ranges for the counts in x and z. Used to define a "pretty" set of values to label legends.
range.cex	The range of expansion for the symbols when this is used to indicate the number of counts in the cells.

xlim	The extreme coordinates in the horizontal direction (see Details).
ylim	The extreme coordinates in the vertical direction (see Details).
red,green,blue	Values in an RGB colorspace to use in transforming the cell values into colors.
alpha	Transparency of the colors.
pch	The symbol to use in displaying the observation density. Either "." or 15 seem to work well depending upon the resolution of the grid.
geocoord	Whether the size of the symbols that indicate density when there are intensity values should be corrected for a Mercator projection.

Details

'densityGrid' expects one matrix or a list of two matrices containing values that will be transformed into colors or sizes of the symbols displayed. The two matrices may be passed as a list. If only one matrix is present, the color of the symbols is determined by the values (counts) in the matrix. If a second matrix is passed, The values in that matrix will be used to determine the colors, and the size of the symbols will be proportional to the values in the first matrix. In the case of only one matrix, the user should set the first value of 'range.cex' to the desired expansion of the symbols.

Currently 'densityGrid' does not display anything in grid cells that have zero count values.

'densityGrid' was developed to allow very large numbers of coordinate locations to be accumulated in a matrix for display on a geographic map. Thus the default limits refer to coordinates as latitude/longitude for the earth. Because some geographic data are so numerous that memory limits are exceeded, the underlying 'makeDensityMatrix' function can be run on small sections of the entire data set and the resulting matrices added as long as the initial coordinate limits are used throughout. Note that if the values for counts (with one matrix) or for intensity (with two matrices) cover a very large range, it may be necessary to trim extreme values (noting this on any legends) and transform the data (usually log10) to get good color separation.

Value

nil. Displays a grid of symbols on an existing plot device.

Author(s)

Jim Lemon

See Also

makeDensityMatrix,color.scale

Examples

```
## Not run:
 data(l2010)
 # log10 transform both density and intensity
 l2010[[1]]<-log10(l2010[[1]])
 l2010[[2]]<-log10(l2010[[2]])
 library(maps)
```

```
x11(width=10)
par(mar=c(7,3,2,3))
plot(0,xlim=c(-180,180),ylim=c(-90,90),type="n",axes=FALSE,xlab="",ylab="")
densityGrid(l2010,pch=".",xrange=c(0,6),zrange=c(2,8),range.cex=c(2,8),
 red=c(0,0.5,1),green=c(0,1,0),blue=c(1,0,0),alpha=1)
color.legend(-60,-107,60,-97,c("2","3","4","5","6","7","8"),
 rect.col=color.scale(1:7,cs1=c(0,0.5,1),cs2=c(0,1,0),cs3=c(1,0,0),alpha=1),
 cex=0.5)
par(xpd=TRUE)
text(0,95,"Lightning strikes 2010")
text(0,-114,"Mean intensity kVA (10^n)",cex=0.5)
points(x=seq(-60,60,20),y=rep(-125,7),pch=".",cex=1:7)
text(x=seq(-60,60,20),y=rep(-132,7),c("<=1","2","3","4","5","6",">6"),cex=0.5)
text(0,-142,"Cell frequency (10^n)",cex=0.5)
par(xpd=FALSE)
map("world",mar=c(7,3,2,3),add=TRUE)
dev.off()
# now only Australia
par(mar=c(7,3,2,3))
plot(0,xlim=c(112,154),ylim=c(-43.8,-11.1),type="n",axes=FALSE,xlab="",ylab="")
densityGrid(l2010,pch=".",xrange=c(0,6),zrange=c(2,8),range.cex=c(2,8),
 xlim=c(112,154),ylim=c(-43.8,-11.1),red=c(0,0.5,1),green=c(0,1,0),
 blue=c(1,0,0),alpha=1)
color.legend(120,-47,146,-45,c("2","3","4","5","6","7","8"),
 rect.col=color.scale(1:7,cs1=c(0,0.5,1),cs2=c(0,1,0),cs3=c(1,0,0),alpha=1),
 cex=0.5)
par(xpd=TRUE)
text(133,-9,"Lightning strikes 2010 (Australia)")
text(133,-48.2,"Mean intensity kVA (10^n)",cex=0.5)
points(x=seq(121,145,4),y=rep(-50,7),pch=".",cex=1:7)
text(x=seq(121,145,4),y=rep(-51,7),c("<=1","2","3","4","5","6",">6"),cex=0.5)
text(133,-52,"Cell frequency (10^n)",cex=0.5)
par(xpd=FALSE)
map("world",mar=c(7,3,2,3),xlim=c(112,154),ylim=c(-43.8,-11.1),add=TRUE)

## End(Not run)
```

`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 lines or capped 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,lty=NA,pch=NA,border=NA,col=par("fg"),display.na=TRUE,
 ...)
```

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.
lty	Line type for redrawing the lines if necessary.
pch	Symbol for redrawing the points if necessary.
border	Line type for drawing a border on the confidence band.
col	Color for the lines or capped bars.
display.na	Whether to display NA values as lines going off the plot.
...	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 pass point and/or line types as in the second example.

The default behavior is to display an undefined dispersion (e.g. a variance with only one observation) as a line going off the plot. If 'display.na' is FALSE, NA values will not be displayed, allowing the user to show only upper or lower dispersion limits.

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

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,pch=1)
disptest2<-matrix(sample(c(TRUE,FALSE),200,TRUE),nrow=10)
disptest.prop<-rowMeans(disptest2)
disptest.ulim<-disptest.llim<-rep(NA,10)
for(i in 1:10) {
 disptest.ulim[i]<-binciWu(disptest2[i,],20)
 disptest.llim[i]<-binciWl(disptest2[i,],20)
}
plot(disptest.prop,main="Dispersion as binomial confidence intervals",
 ylim=c(min(disptest.llim),max(disptest.ulim)),
 xlab="Sample",ylab="Proportion")
dispersion(1:10,disptest.prop,disptest.ulim,disptest.llim,
 interval=FALSE,lty=2,pch=1)
```

`do.first` *Execute a graphic function on a plot*

Description

do.first allows the user to execute one or more commands before displaying values on a plot.

Details

'`do.first`' is an argument in some plotrix functions that allows the user to do things like add a background color or a grid to the plot before displaying the other plot elements.

The value of '`do.first`' should be a character string that can be parsed to one or more valid R commands. Remember to enclose the string in quotes, separate commands with semicolons and escape quotes within the string with backslashes if necessary.

`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>
http://www.stat.auckland.ac.nz/~rolf

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=0.05,col=NA,lwd=NA,...)
```

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.
`lwd`	Line width for the arc.
`...`	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. For argument 'n' (which may be either a scalar or a vector, although most likely you will leave it at the default value), an integer value means to use that number of segments to approximate the arc, while a non-integer value means to use enough segments so that the angle that successive segments make with one another is no more than n radians.

Value

Returns a matrix of expanded arguments invisibly.

Author(s)

Gabor Grothendieck. Improvements by Ted Toal.

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)
draw.arc(5, 5, 3, deg1=100, deg2=170, col="gold", lwd=50, lend=1)
# 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,density=NULL,
 angle=45,lwd=1)
```

Arguments

x,y	Coordinates of the center of the circle.
radius	Radius (or radii) of the circle(s) 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.
density	Density for patterned fill. See 'polygon'.
angle	Angle of patterned fill. See 'polygon'.
lwd	Line width for the circumference.

Details

'draw.circle' uses the dimensions of the plot and the 'x' and 'y' coordinates to draw a circle rather than an ellipse.

Value

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

Author(s)

Jim Lemon, thanks to David Winsemius for the density and angle args

See Also

polygon

Examples

```
plot(1:5,seq(1,10,length=5),type="n",xlab="",ylab="",main="Test draw.circle")
draw.circle(2,4,c(1,0.66,0.33),border="purple",
 col=c("#ff00ff","#ff77ff","#ffccff"),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",col="yellow",lty=1,
 density=5,angle=30,lwd=10)
draw.circle(3.5,8,0.8,border="blue",lty=2,lwd=2)
```

`draw.ellipse` *Draw ellipse*

Description

Draws ellipses on an existing plot.

Usage

```
draw.ellipse(x, y, a = 1, b = 1, angle = 0, segment = NULL,
arc.only = TRUE, deg = TRUE, nv = 100, border = NULL,
col = NA, lty = 1, lwd = 1, ...)
```

Arguments

`x`	A vector or a matrix (if `y` is missing).
`y`	A vector, can be missing.
`a,b`	Vectors, radii of the ellypses along the two axes in user units.
`angle`	Angle of rotation in degrees (if `deg=TRUE`) or in radians (if `deg=FALSE`).
`segment`	Start and endpoints of arc in degrees (if `deg=TRUE`) or in radians (if `deg=FALSE`).
`arc.only`	Logical, if `segmen` the full ellipse is not drawn, radii from the ends of the arc are drawn to form a sector (see Examples).
`deg`	Logical, if angles are given in degrees (`TRUE`) or radians.
`nv`	Number of vertices to draw the ellipses.
`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.
`...`	Additional arguments passed to `polygon`.

Value

Draw ellipses as a side effect.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

polygon

Examples

```
plot(c(0,10), c(0,10), type="n", main="test draw.ellipse")
draw.ellipse(c(3,7), c(8,8), c(0.5,1), c(1,0.5), col=c(2,4),
 angle=c(45,0), segment=rbind(c(0,45),c(45,360)))
draw.ellipse(c(3,7), c(6,6), c(0.5,1), c(1,0.5), col=c(2,4),
 angle=c(45,0), segment=rbind(c(0,45),c(45,360)), arc.only=FALSE)
draw.ellipse(c(3,7), c(4,4), c(0.5,1), c(1,0.5), border=c(2,4),
 angle=c(45,0), segment=rbind(c(0,45),c(45,360)), arc.only=FALSE)
draw.ellipse(c(3,7), c(2,2), c(0.5,1), c(1,0.5), border=1,
 angle=c(45,0), lty=3)
draw.ellipse(c(3,7), c(2,2), c(0.5,1), c(1,0.5), border=c(5,3),
 angle=c(45,0), nv=c(3,4), lty=2, lwd=2)
```

---

draw.radial.line *Draw a radial line*

Description

Draws a line radiating from a specified center, optionally expanding the line width as a function of distance from center.

Usage

```
draw.radial.line(start, end, center=c(0, 0), angle=0, deg=NA,
 expand=FALSE, col=NA, lwd=NA, ...)
```

Arguments

start	Distance from center of circular area to start of line in x/y user units.
end	Distance from center of circular area to end of line in x/y user units.
center	The center of the circular area in x/y user units.
angle	The angular position of the line in radians.
deg	The angular position of the line in degrees (takes precedence if not NA).
expand	TRUE to expand line width in proportion to distance from center.
col	The color of the line, NA for par("col").
lwd	The width of the line in device-specific units, NA for par("lwd").
...	Arguments passed to 'lines' (expand=FALSE) or 'polygon' (expand=TRUE).

Details

If the user passes a value for 'deg', this overrides any value passed to 'angle'.

If 'expand' is FALSE, the line width is constant (as specified by par("lwd").

If 'expand' is TRUE, the line width is equal to the lwd value at distance 'end' and contracts as it moves towards 'start'. When expand is 'TRUE', lty is ignored.

Value

nil

Author(s)

Ted Toal

See Also

lines, linkdraw.arc

Examples

```
plot(0, xlim=c(1,5), ylim=c(1,5), main="Test of radial lines", xlab="", ylab="", type="n")
points(3, 3, pch=20)
draw.radial.line(1, 2, center=c(3,3))
draw.radial.line(1, 2, center=c(3,3), angle=pi/4)
draw.radial.line(1, 2, center=c(3,3), angle=pi/4+0.1, col="blue", lwd=4, lty=3)
draw.radial.line(0.2, 1.2, center=c(3,3), deg=120, col="red", lwd=10)
draw.radial.line(0.2, 1.2, center=c(3,3), deg=145, col="purple", lwd=10, lend=1)
draw.radial.line(0.5, 2, center=c(3,3), deg=225, expand=TRUE, col="gold")
draw.radial.line(0.7, 1.4, center=c(3,3), deg=180, expand=TRUE, col="orange", lwd=30)
draw.radial.line(0.5, 1.5, center=c(3,3), deg=235, expand=TRUE, lwd=5, col="brown")
draw.radial.line(0.1, 1.5, center=c(3,3), deg=325, expand=TRUE, lwd=5, col="green")
```

---

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

Description

Displays a 3D pie sector.

Usage

```
draw.tilted.sector(x=0,y=0,edges=NA,radius=1,height=0.1,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.

If 'explode' is zero, only the top and outer side of each sector will be displayed. This will sometimes fix the problem of a pie with one huge sector greater than 3*pi/2 that cannot otherwise be drawn.

Value

The bisector of the pie sector in radians.

Author(s)

Jim Lemon

See Also

pie3D

---

drawNestedBars *Display nested bars*

Description

Displays the nested bars for barNest.

Usage

```
drawNestedBars(x,start,end,shrink=0.1,errbars=FALSE,intervals=TRUE,col=NA,
labelcex=1,lineht=NULL,showall=TRUE,Nwidths=FALSE,barlabels=NULL,
showlabels=TRUE,arrow.cap=NA)
```

Arguments

x	One level of the breakdown produced by 'brkdnNest'.
start,end	The left and right x coordinates for the bar or group of bars to be displayed.
shrink	The proportion to shrink the width of the bars at each level.
errbars	Whether to display error bars on the bars.
intervals	Whether to use offsets or absolute values when displaying measures of dispersion.
col	The colors to use to fill the bars. See Details.
labelcex	Character size for the group labels.
lineht	The height of a margin line in user units.
showall	Whether to display the bars at any levels above the last.
Nwidths	Whether to scale the widths of the bars to the number of observations.
barlabels	Optional labels to display below the bars.
showlabels	Whether to display the labels below the bars.
arrow.cap	The width of the "cap" on error bars in user units, defaulting to 0.01 of the width of the plot.

Details

'drawNestedBars' displays the bars for the nested breakdown performed by 'brkdnNest'. It starts at the top of the list and calls itself for each level of the breakdown. It is unlikely to be useful for anything else.

The combination of 'showlabels=TRUE' and 'showall=FALSE' allows the display of all of the labels below the plot with only the last set of bars being displayed. To have a set of labels not displayed, pass explicit 'barlabels' and have zero length labels for the level that is not to have labels.

Value

nil

Author(s)

Jim Lemon and Ofir Levy

See Also

brkdnNest, drawNestedBars

`drawSectorAnnulus` *Display a radial pie sector*

Description

Displays a radial pie sector with optional annuli.

Usage

```
drawSectorAnnulus(angle1,angle2,radius1,radius2,col,angleinc=0.03)
```

Arguments

`angle1,angle2`	Start and end angle for the sector.
`radius1,radius2`	Start and end of the radial extent of the annulus.
`col`	Color of the sector.
`angleinc`	The angular increment to use in drawing the arcs.

Details

'`drawSectorAnnulus`' displays a single radial pie sector. It is probably only useful when called from radial.pie.

Value

nil

Author(s)

Jim Lemon

See Also

radial.pie

ehplot *Engelmann-Hecker-Plot - EH-Plot*

Description

This R function provides a convenient way to visualize the distribution of grouped numerical data.

Usage

```
ehplot(data, groups, intervals=50, offset=0.1, log=FALSE,
       median=TRUE, box=FALSE, boxborder="grey50",
       xlab="groups", ylab="values", col="black",
       add=FALSE, sort=TRUE, ...)
```

Arguments

data	Vector of numerical data.
groups	Vector of group names which should have the same length as data.
intervals	The data is splitted into a certain number of intervals. If data points are in the same interval they are drawn side-by-side.
offset	x-distance between two data points at the same interval.
log	Logarithmic display
median	To show the median of each group. NAs in data are not considered for calculating the medians.
box	To underlay a boxplot.
boxborder	The color of the boxplot if a boxplot is drawn.
xlab	x-axis label
ylab	y-axis label
col	vector of colors for the datapoints. (recycled as necessary).
add	add this plot to an existing one (i.e. do not call plot.new).
sort	normally, the groups are sorted by name. To keep the order as provided in the groups-vector, set this to FALSE
...	additional plot-parameters will be passed to the plot-function

Author(s)

Robby Engelmann <robby.engelmann@med.uni-rostock.de> and Michael Hecker <michael.hecker@rocketmail.com>

Examples

```
data(iris)
ehplot(iris$Sepal.Length, iris$Species, intervals=20, cex=1.8, pch=20)
ehplot(iris$Sepal.Width, iris$Species, intervals=20, box=TRUE, median=FALSE)
ehplot(iris$Petal.Length, iris$Species, pch=17, col="red", log=TRUE)
ehplot(iris$Petal.Length, iris$Species, offset=0.06, pch=as.numeric(iris$Species))

# Groups don't have to be presorted:
rnd <- sample(150)
plen <- iris$Petal.Length[rnd]
pwid <- abs(rnorm(150, 1.2))
spec <- iris$Species[rnd]
ehplot(plen, spec, pch=19, cex=pwid, col=rainbow(3, alpha=0.6)[as.numeric(spec)])
```

---

election *Assign party members to seats*

Description

Create a layout for an election result in an assembly

Usage

```
election(seats,result,formula,colours = sample(rainbow(length(counts))))
```

Arguments

seats	A data frame of x and y positions, row numbers and angles (usually the output from the seats function).
result	A data frame with party names and seat counts.
formula	A formula with the party name column on the left and the count column on the right. Think of the twiddle symbol as "got".
colours	A vector of colours. If missing a random rainbow is used. This may cause Green parties to show as red.

Value

A data frame including:

x	The x positions of the seats to be plotted on semi-circular arcs.
y	The y positions of the seats to be plotted on semi-circular arcs.
r	The row numbers for each seat.
theta	The angle of each seat, going from pi to zero radians.
party	The labels for the party holding each seat.
colour	The colour that has been assigned to the party.

Author(s)

Barry Rowlingson

See Also

seats

Examples

```
# The EU parliament has 751 seats, and Wikipedia currently shows this
eu = structure(list(colour = c("#3399FF", "#F0001C", "#0054A5", "#FFD700",
"#990000", "#909090", "#32CD32", "#40E0D0"), party = c("EPP",
"S and D", "ECR", "ALDE", "GUE-NGL", "Non-Inscrits", "Greens-EFA",
"EFDD"), members = c(220L, 191L, 70L, 68L, 52L, 52L, 50L, 48L
)), .Names = c("colour", "party", "members"), row.names = c(NA,
-8L), class = "data.frame")
strasbourg = seats(751, 16)
eugov = election(strasbourg, eu, party~members, colours=eu$colour)
oldmar<-par(mar=c(2,4,4,2))
plot(eugov$x, eugov$y, col=eugov$colour, asp=1, pch=19, ylim=c(-2,2.5),
 xlab="", ylab="", main="EU Parliament 2014", axes=FALSE)
legend(-0.7,-0.3,eu$party,fill=eu$colour)
par(oldmar)
# or using ggplot2
## Not run:
 require(ggplot2)
 blank = theme(axis.line=element_blank(),
       axis.text.x=element_blank(),
 axis.text.y=element_blank(),
 axis.ticks=element_blank(),
 axis.title.x=element_blank(),
 axis.title.y=element_blank(),
 panel.background=element_blank(),
 panel.border=element_blank(),
 panel.grid.major=element_blank(),
 panel.grid.minor=element_blank(),
 plot.background=element_blank())
 ggplot(eugov, aes(x=x,y=y,col=party)) + geom_point() + coord_fixed() + blank

## End(Not run)
```

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

Description

Try to find the largest empty rectangle on a plot.

Usage

```
emptyspace(x,y=NULL)
```

Arguments

x,y	x and y positions of the points on the plot.

Details

'emptyspace' searches the pairs of points on the plot to find the largest rectangular space within which none of the points lie. It does not guarantee that the space will be large enough to fit a legend or text.

Two alternatives are the 'largest.empty' function in the **Hmisc** package and the 'maxEmptyRect' function. While 'maxEmptyRect' will generally outperform 'emptyspace', 'emptyspace' will sometimes find a slightly smaller, but "squarer" rectangle.

Value

The 'x' and 'y' coordinates of the center of the rectangle found.

Author(s)

Ray Brownrigg

Examples

```
x<-rnorm(100)
y<-rnorm(100)
plot(x,y,main="Find the empty space",xlab="X",ylab="Y")
es<-emptyspace(x,y)
# use a transparent background so that any overplotted points are shown
boxed.labels(es,labels="Here is the\nempty space",bg="transparent")
```

---

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

'fan.plot' displays sectors much like a pie chart except that the sectors are overlapped. this allows the angular extents of the sectors to be visually compared much more accurately by the viewer. Sectors are plotted from the largest to the smallest, shrinking the radius of each successive sector.

When sending output to the postscript device, the resulting image can be trimmed by changing the values in BoundingBox in the header with a text editor.

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

Examples

```
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$threatened,max.span=pi,
 labels=paste(iucn.df$area,iucn.df$threatened,sep="-"),
 main="Threatened species by geographical area (fan.plot)",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.

Feather plots are typically wider than high. The user will probably want to specify a graphics device that doesn't leave lots of blank space above and below the plot.

Value

nil

Author(s)

Jim Lemon, Eduardo Klein

See Also

spread.labels

Examples

```
dev.new(width=8,height=3)
r<-0.6+rnorm(24)/5
theta<-c(seq(15*pi/16,pi/16,length.out=12),
 seq(17*pi/16,31*pi/16,length.out=12))
feather.plot(r,theta,xlabels=1:24,
 main="Standard Coordinates",xlab="Time",ylab="Value")
# rearrange theta for meteorological coordinates
feather.plot(r,c(theta[19:24],rev(theta[7:18]),theta[1:6]),xlabels=1:24,fp.type="m",
 main="Meteorological Coordinates",xlab="Time",ylab="Value")
dev.off()
```

`fill.corner` *Fill a "corner" of a matrix with values*

Description

Fills one corner of a matrix with the supplied values, leaving the rest filled with a default value.

Usage

```
fill.corner(x,nrow,ncol,na.value=NA)
```

Arguments

`x`	A vector of values.
`nrow,ncol`	The number of rows and columns in the matrix to be returned.
`na.value`	The default value for unfilled cells.

Details

'`fill.corner`' creates an nrow by ncol matrix and fills the lower left corner with the values supplied in '`x`'. If there are more values in '`x`' than cells in the matrix, only the first nrow*ncol values will be inserted.

Value

An nrow by ncol matrix containing the values in '`x`'.

Author(s)

Jim Lemon

find_max_cell *Maximum (or minimum) value cell in a matrix.*

Description

Find the indices of the maximum value in a matrix.

Usage

```
find_max_cell(x,max=TRUE)
```

Arguments

x	a numeric matrix
max	The default is to return the indices of the maximum value(s). 'max=FALSE' returns those of the minimum.

Value

A list containing the column (x) and row (y) indices.

Note

Intended to enable the user to mark cells in sampcolor2D.matplot. Remember to subtract 0.5 from both values to center the mark in the cell.

Author(s)

Jim Lemon

floating.pie *Display a floating pie chart*

Description

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

Usage

```
floating.pie(xpos=0,ypos=0,x,edges=200,radius=1,col=NULL,startpos=0,
 shadow=FALSE,shadow.col=c("#ffffff","#cccccc"),explode=0,...)
```

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
shadow.col	Colors to use for a shadow.
explode	How much to "explode" one or more of the sectors.
...	graphical parameters passed to 'polygon'

Details

'floating.pie' displays a pie chart with an optional shadow on an existing plot (see 'polygon.shadow'). 'floating.pie' now accepts NAs or zeros in 'x', but simply ignores them.

'floating.pie' can be useful when multiple pie charts are placed on a plot overlaying something else, like a map.

Value

The bisecting angle of the sectors in radians. Useful for placing text labels for each sector. If any values in 'x' were zero or NA, no angle is returned for that value. This means that the user must adjust the labels accordingly if 'pie.labels' is called.

If 'floating.pie' is called with no graphics device, it will try to open one with the appropriate dimensions.

If 'pie.labels' is called, ensure that the center of the pie chart and any 'explode' values are the same.

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"))
plot(0,xlim=c(-1.5,1.5),ylim=c(-1.5,1.5),type="n",axes=FALSE,
 main="Floating pie with minor explosions",xlab="",ylab="")
floating.pie(x=1:5,explode=c(0,0.1,0,0.2,0))
```

`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 part of ...) 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",col.axis="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])
```

---

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,
 label.cex=1,border.col=NA,priority.label="Priorities",
 priority.extremes=c("High","Low"),time.axis=3)
```

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 (see Details).
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.
label.cex	Relative size for the task labels at the left side.
border.col	The color for an optional border for the bars (NA=none).
priority.label	Label for the priority color legend.
priority.extremes	Labels for each end of the priority color legend.
time.axis	Where to place the time axis labels.

Details

Because the "time" axis is calculated using POSIXct values, the values passed as 'xlim' must also be POSIXct. See the second plot in the examples.

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. If colors other than 'rainbow' are wanted, remember to pass enough colors for one to the lowest (highest numerically) priority.

There can now be more than one time interval for each task. If there is, more than one bar will be displayed for each interval, which may not be a task at all, but rather intervals related to the labels.

Colors can be specified for labels or intervals and if there are not as many colors as intervals, the first "number of unique labels" colors will be assigned to each unique label. This should make every bar for each label the same color, but be aware that the colors will be distributed in alphabetical order of the entity labels. If there are at least as many taskcolors as intervals, they will be assigned to intervals in the order of the 'taskcolors' vector. The examples should make this clearer.

Since 'gantt.chart' can be used to display things other than prioritized tasks, the labels for the priority legend can now be specified.

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 - added label colors by Nicolas Immelman)

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)
# add a little extra space on the right side
gantt.chart(gantt.info,main="Calendar date Gantt chart (2004)",
priority.legend=TRUE,vgridpos=vgridpos,vgridlab=vgridlab,hgrid=TRUE,
xlim=as.POSIXct(strptime(c("2004/01/01","2004/12/20"),
format=Ymd.format)))
# 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="All bars the same color",taskcolors="lightgray")
gantt.chart(info2,vgridlab=8:19,vgridpos=8:19,
 main="A color for each label",taskcolors=c(2,3,7,4,8))
gantt.chart(info2,vgridlab=8:19,vgridpos=8:19,
 main="A color for each interval - with borders",
 taskcolors=c(2,3,7,4,8,5,3,6,"purple"),border.col="black")
```

`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,xlim=NA,ylim=NA,xlab=NULL,
 ylab=NULL,horiz=FALSE,col,...)
```

Arguments

`y`	a vector of 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
`xlim`	Optional x limits for the plot
`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,xlim=NA,ylim=NA,plot=TRUE,border=par("fg"),col=NULL,log="",
axis.labels=NULL,axes=TRUE,pars=list(boxwex=0.8,staplewex=0.5,outwex=0.5),
horizontal=FALSE,add=FALSE,at=NULL,main=NULL,xlab="",ylab="")
```

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
xlim,ylim	Optional x and 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.
axes	Whether to display axes.
pars	optional parameters for consistency with 'boxplot'
horizontal	whether to plot horizontal boxplots - currently does nothing
add	whether to add the boxplot(s) to an existing plot - currently does nothing.
at	optional horizontal locations for the boxplots.
main	a title for the plot.
xlab,ylab	X and Y axis labels.

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. For example, in the second example, the value returned for the upper staple of the right boxplot is 14 rather than 20, due to the 6 unit gap.

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=range(x),ylim=range(y),xticlab,xtics=NA,yticlab,ytics=NA,
 lty=rep(1,length(x)),col=rep(par("col"),length(x)),pch=rep(1,length(x)),
 add=FALSE,stax=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.
stax	whether to call staxlab for staggered axis labels.
...	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 may 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, do.first

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(0,42),
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")
```

---

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

Description

Displays a barplot with a missing range.

Usage

```
gap_barp(height,gap,width=0.4,names.arg=names(height),
col=NULL,main="",xlab="",ylab="",xlim=NULL,ylim=NULL,x=NULL,
height.at=pretty(height),height.lab=NULL,...)
```

Arguments

height	a vector of data values
gap	the range of values to be left out
width	the proportion of bar width to bar spacing divided by 2. width=1 means no spaces between the bars.
names.arg	labels for the bars.
col	color(s) in which to plot the values
main	title for the plot.
xlab	label for the x axis
ylab	label for the y axis
xlim	Optional x limits for the plot
ylim	optional y limits for the plot
x	optional x positions for the bars.
height.at	explicit positions for the y axis ticks
height.lab	explicit labels for the y axis ticks.
...	arguments passed to 'barp'.

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 or lower limit.

Value

The center positions of the bars.

Author(s)

Jim Lemon

See Also

barp

Examples

```
oneout<-c(rnorm(5,sd=5),20,rnorm(5,sd=5))
gap_barp(oneout,gap=c(8,16),xlab="Index",height.at=c(-5,0,5,20),
 ylab="Group values",main="Barplot with gap above zero")
oneout[6]<--20
gap_barp(oneout,gap=c(-8,-16),xlab="Index",height.at=c(-20,-5,0,5),
 ylab="Group values",main="Barplot with gap below zero")
```

---

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.tablepos` *Get the position for a legend or table*

Description

Gets the x and y positions and justification for a legend or table in user units from the string descriptors like "top".

Usage

```
get.tablepos(x)
```

Arguments

x	A valid position descriptor like "top".

Details

'`get.tablepos`' checks for one of the nine valid position descriptors:

topleft, top, topright, left, center, right, bottomleft, bottom and bottomright.

If none of these descriptors are found, it will return the center position and justification.

Value

A list containing:

x	x position
y	y position
xjust	x (horizontal) justification
yjust	y (vertical) justification

Author(s)

Jim Lemon

See Also

addtable2plot, legendg

`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 *Get coordinates in the figure region in user units.*

Description

Calculates the coordinates of a proportional point of the figure region in user units.

Usage

```
getFigCtr(pos=c(0.5,0.5))
```

Arguments

pos	The proportion of the figure region to find (see Details).

Details

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

By changing 'pos', any proportional points of the figure region can be returned. For example, 'pos=c(0,0)' will return the left and bottom coordinates of the figure region.

Value

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

Author(s)

Jim Lemon (thanks to Karl Brand for the adjustable coordinates)

getIntersectList *Enter a set intersection list*

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.

It is very helpful when there are large numbers of elements, as the 'makeIntersectList' function runs very slowly.

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))
newmarinfo<-getMarginWidth(labels=c("Long label","Even longer label"))
oldmar<-par("mar")
par(mar=c(oldmar[1:3],newmarinfo$newmar))
plot(rnorm(10))
par(xpd=TRUE)
text(rep(newmarinfo$marcenter,2),c(0.5,-0.5),
 c("Long label","Even longer label"))
par(mar=oldmar,xpd=FALSE)
```

getYmult *Correct for aspect and coordinate ratio*

Description

Calculate a multiplication factor for the Y dimension to correct for unequal plot aspect and coordinate ratios on the current graphics device.

Usage

```
getYmult()
```

Details

'getYmult' retrieves the plot aspect ratio and the coordinate ratio for the current graphics device, calculates a multiplicative factor to equalize the X and Y dimensions of a plotted graphic object.

Value

The correction factor for the Y dimension.

Author(s)

Jim Lemon

See Also

draw.circle

get_axispos3d *Get axis positions on a 3D plot*

Description

Calculate the axis positions on a 3D plot.

Usage

```
get_axispos3d(edge,pmat,at,pos=NULL, dist=0)
```

Arguments

edge	which axis to calculate.
pmat	matrix to transform coordinates.
at	position on the axis.
pos	position of the axis relative to the other axes.
dist	Offset of the axis.

Value

A position in 2D coordinates

Author(s)

Ben Bolker

`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

`histStack` *Histogram "stacked" by categories*

Description

Histogram of a quantitative variable with bars that are "stacked" by the values of a factor variable.

Usage

```
 histStack(x,...)

 ## S3 method for class 'formula'
histStack(x,data,breaks="Sturges",col="rainbow",
  right=TRUE,main="",xlab=NULL,legend.pos=NULL,cex.legend=0.75,...)

 ## Default S3 method:
histStack(x,z,breaks="Sturges",col="rainbow",
  right=TRUE,main="",xlab=NULL,legend.pos=NULL,cex.legend=0.75,...)
```

Arguments

`x`	A vector of quantitative data or a formula of the form x~z (see z below).
`z`	A vector of categorical data (a factor) that will define the “stacks”.
`data`	A data frame that contains both x and z.
`breaks`	Breaks to use in categorizing values of x.
`col`	Either a vector of colors in any legitimate form or a character string that specifies a function that requires only the length of the vector as an argument and will return a vector of colors with that length. (see Details)
`right`	A logical that indicates whether the bins are right-open (left-closed; =TRUE) or right-closed (left-open; =FALSE; default).
`main`	A character string that forms the main title for the plot.
`xlab`	A character string for labeling the x-axis.
`legend.pos`	A character string or two numeric values indicating the position for the stacking legend.
`cex.legend`	A numeric character expansion value for the legend. Values less than 1 will make the legend smaller.
`...`	Additional arguments sent to the hist function.

Details

'histStack' displays a "stacked histogram" while using many of the same arguments as hist(). The argument 'z' will be converted to a factor with a warning if it is not already a factor.

The color functions in **grDevices** (e.g. "gray.colors") should always be valid when passed as the 'col' argument. Any function that will return a vector of 'n' colors when called with a single argument 'n' and that exists in the current environment should work. An error will occur if length(col)==1 and the value is not a function as described for 'col' (e.g., 'col="blue"' will result in an error).If fewer colors than levels of 'z' are passed, they will be recycled.

Value

nil. A plot is displayed.

Note

This function is currently experimental.

Author(s)

Derek Ogle with modifications by Jim Lemon

See Also

hist, legend

Examples

```
set.seed(409)
df<-data.frame(len=rnorm(100)+5,
 grp=sample(c("A","B","C","D"),100,replace=TRUE))
histStack(len~grp,data=df,main="Default (rainbow) colors",
 xlab="Length category")
histStack(len~grp,data=df,col="heat.colors",main="Heat colors",
 xlab="Length category",legend.pos="topright")
histStack(len~grp,data=df,col=2:5,main="Colors by number",
 xlab="Length category",legend.pos=c(2.8,18))
```

---

intersectDiagram *Display set intersections*

Description

Display set intersections as rows of rectangles.

Usage

```
intersectDiagram(x,pct=FALSE,show.nulls=FALSE,xnames=NULL,sep="+",
 mar=c(0,0,3,0),main="Intersection Diagram",cex=1,col=NULL,
 minspacing=NA,all.intersections=FALSE,include=NULL,null.label="Non-set")
```

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 or a two column matrix of object identifiers and attribute identifiers 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 are not members of any set. Any value that is not NA will become the label for this category.
xnames	Optional user supplied names for the set categories (see Details).
sep	The separator to use between category names (see Details).
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).
cex	Character expansion for the intersection labels.
minspacing	The minimum spacing between the rectangles (see Details).
all.intersections	Whether to display all intersections, even if empty (Dangerous - see Detail).
include	Which set identifiers to include in the diagram (see Details).
null.label	The label for the non-set entities if displayed.

Details

'intersectDiagram' displays rows of optionally colored rectangles that represent the intersections of set memberships (attributes) of a set of objects. The topmost row represents the intersections of the fewest sets, and succeeding rows represent the intersections of more sets. 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.

Important - If the 'all.intersections' argument is TRUE, all intersections will be displayed, whether empty or not (see the example). This is mostly for demonstration purposes, and if the number of sets is large, is likely to produce a very messy diagram. Similarly, sets with large numbers of intersections that are populated will require very large displays to be readable, even if there are small numbers in the intersections. If you would like to see this in action, pass the data frame 'setdf' in the categoryReshape example to 'intersectDiagram' with 'all.intersections' TRUE.

'intersectDiagram' does not attempt to display the set intersections as a pattern of overlapping geometric figures, but rather the relative numbers of objects sharing each intersection. More than three intersecting sets generally produce a complex and difficult to interpret Venn diagram, and this provides an alternative way to display the size of intersections between larger numbers of sets.

'intersectDiagram' now allows the user to display only part of the set intersections, which is useful for analyzing very complex intersections. This is controlled by the 'include' argument. This defaults to all sets or attributes when 'include=NULL'. If one or more of the labels of the

sets or attributes is passed, only the intersections containing those labels will be displayed. See examples 2 and 3 below.

Each set (attribute) 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 intersectList object passed as 'x' or returned from the call to 'makeIntersectList'.

If a matrix or data frame of set membership indicators is passed as 'x', it will be passed to makeIntersectList for conversion. Each column must represent a set, and the values in the columns must be 0 or 1, or FALSE or TRUE. Similarly, if a matrix or data frame in which the first column is object identifiers and the second column is attributes, this will be passed to 'makeIntersectList'.

The spacing between the largest rectangles is controlled by 'minspacing'. 'minspacing' is in units of object counts and defaults to 0.1 times the largest number of objects in an intersection. When the number of objects in different intersections at a given level varies widely, the labels of intersections with few objects may overlap if they are wide relative to the rectangle representing the number of objects. This can be corrected by passing a 'minspacing' argument that will increase the space between rectangles and/or decreasing the character size of the labels. If the labels for each set are relatively long, setting 'namesep="\n"' may help. Note that if a different separator is passed, that separator must be explicitly passed in any subsequent calls using the same 'intersectList' object - see examples 1 to 3 below.

Value

Returns the intersectionList object invisibly.

Author(s)

Jim Lemon

See Also

makeIntersectList, getIntersectList, categoryReshape

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,prob=c(0.15,0.85)),
 sample(c(0,1),200,TRUE,prob=c(0.35,0.65)),
 sample(c(0,1),200,TRUE,prob=c(0.5,0.5)),
 sample(c(0,1),200,TRUE,prob=c(0.9,0.1))),ncol=4)
colnames(druguse)<-c("Alc","Tob","THC","Amp")
druglist<-makeIntersectList(druguse,sep="\n")
# first display it as counts
intersectDiagram(druglist,main="Patterns of drug use",sep="\n")
# then display only the intersections containing "Alc"
intersectDiagram(druglist,main="Patterns of drug use (Alcohol users only)",
 sep="\n",include="alc")
```

```
# now display only the intersections containing "Amp"
intersectDiagram(druglist,main="Patterns of drug use (Speed users only)",
 sep="\n",include="amp")
# then as percent with non.members, passing the initial matrix
intersectDiagram(druguse,pct=TRUE,show.nulls=TRUE)
# alter the data to have more multiple intersections
druguse[which(as.logical(druguse[,1]))[1:40],2]<-1
druguse[which(as.logical(druguse[,1]))[31:70],3]<-1
druguse[,4]<-sample(c(0,1),200,TRUE,prob=c(0.9,0.1))
intersectDiagram(druguse,main="Smaller font in labels",
 col=c("gray20","gray40","gray60","gray80"),cex=0.8)
# transform the spacing - usually makes it too close, first try minspacing
intersectDiagram(druguse,col="gray",main="Minimum spacing = 30 cases",
 minspacing=30)
# then try cex - may need both for large differences
intersectDiagram(druguse,main="Very boring single color",col="gray",cex=0.8)
# create a matrix with empty intersections
druguse<-matrix(c(sample(c(0,1),20,TRUE),
 sample(c(0,1),20,TRUE),
 sample(c(0,1),20,TRUE),
 sample(c(0,1),20,TRUE)),ncol=4)
# show only the populated intersections
intersectDiagram(druguse,main="Display only populated intersections")
# show all intersections
intersectDiagram(druguse,main="Display empty intersections",all.intersections=TRUE)
```

`jiggle` *Calculate equally spaced values within a range.*

Description

Calculates a specified number of equally spaced values in a range

Usage

```
jiggle(n,range=c(-1,1))
```

Arguments

`n`	The number of values to calculate.
`range`	The range within which to fit the values.

Details

'`jiggle`' is an alternative to the '`jitter`' function. Instead of using '`runif`' to provide the values, it calls '`sample`' and then scales the resulting values to the range specified. This guarantees that the values will be evenly spaced.

Value

A vector of n values within the range specified.

Author(s)

Jim Lemon

Examples

```
ahw.df<-data.frame(Age=rnorm(100,35,10),
 Height=rnorm(100,160,15),Weight=rnorm(100,75,20))
par(mfrow=c(1,3))
boxplot(ahw.df$Age,main="Age")
points(jiggle(100,c(0.5,1.5)),ahw.df$Age,col="red")
boxplot(ahw.df$Height,main="Height")
points(jiggle(100,c(0.5,1.5)),ahw.df$Height,col="green")
boxplot(ahw.df$Weight,main="Weight")
points(jiggle(100,c(0.5,1.5)),ahw.df$Weight,col="blue")
```

---

joyPlot *Display a series of density curves.*

Description

'joyPlot' displays a matrix of density curves or other two component lists whose names are 'x' and 'y'. The labels for each line/polygon are displayed on the left axis of the plot. The labels default to the names of the components of 'x' if these are present.

Usage

```
joyPlot(x,mar=c(5,4,4,2),newrange=c(0,1),border=NA,fill=NULL,
main="",xlab="",ylab="",xlim=NA,line_labels=names(x),xat=NULL,
xaxlab=NULL)
```

Arguments

x	A list of density curves or other objects with x and y values.
mar	Margins for the plot.
newrange	Passed to 'rescale' to scale the values to fit the bands on the plot. See Details.
border	The border colors for the polygons.
fill	Optional fill colors for the polygons.
main	Text for the title for the plot.
xlab,ylab	The x and y axis labels.
xlim	Optional limit for the x axis as 'density' returns values outside the range of the values in 'x'.
line_labels	Labels for the lines/polygons dieplayed.
xat	Optional custom x tick positions.
xaxlab	Optional custom x tick labels.

Details

The density curves or other x/y lists will be scaled so that the largest will fit into the one user unit band allocated for each curve by default. If the second value of 'newrange' is changed, the heights of the curves will change proportionately. See the third exampls.

Value

nil

Author(s)

Jim Lemon

See Also

plot, stackpoly

Examples

```
x1<-c(sample(20:50,20),sample(40:80,30))
x2<-c(sample(10:40,30),sample(50:90,30))
x3<-sample(20:90,50)
xdens1<-density(x1)
xdens2<-density(x2)
xdens3<-density(x3)
joyPlot(list(xdens1,xdens2,xdens3),main="joyPlot with lines",
 xlab="Position",xlim=c(0,100))
xlist<-list(first=xdens1,second=xdens2,third=xdens3)
joyPlot(xlist,main="joyPlot with polygons",xlab="Position",
 fill=c("#ffcccc","#ccffcc","#ccccff"),xlim=c(0,100))
joyPlot(xlist,main="joyPlot with overlapping polygons",
 fill=c("#ffcccc","#ccffcc","#ccccff"),xlim=c(0,100),
 newrange=c(0,1.5),xlab="Position")
```

---

kiteChart *Magnitude by position chart.*

Description

Display numeric values as the widths of a polygon along a dimension such as time.

Usage

```
kiteChart(x,xlim=NA,ylim=NA,timex=TRUE,main="Kite chart",
xlab=ifelse(timex,"Time","Groups"),ylab=ifelse(timex,"Groups","Time"),
border=par("fg"),col=NULL,varpos=NA,varlabels=NA,varscale=FALSE,
timepos=NA,timelabels=NA,mar=c(5,4,4,4),axlab=c(1,2,3,4),
normalize=FALSE,shownorm=TRUE,...)
```

Arguments

x	Numeric matrix or data frame
xlim	Horizontal extent of the chart. Defaults to 1:dim(x)[2].
ylim	Vertical extent of the chart. Defaults to 0.5:dim(x)[1]+0.5.
timex	Whether the "time" axis is x (horizontal) or not.
main,xlab,ylab	As in 'plot'.
border	The border color(s) for the polygons.
col	The fill colors for the polygons.
varpos	Optional positions for the "kite lines". Defaults to 1:dimx[1]. (see Details)
varlabels	Labels for the rows of values - defaults to the rownames, or if these are missing, varpos[1:dim(x)[1]].
varscale	Whether to show the maximum extent of each "kite line".
timepos	The positions of the values along the x axis, usually times, defaulting to 1:dim(x)[2].
timelabels	Labels for the positions, defaulting to 'timepos'.
mar	Plot margins. These leave space for the normalization multipliers on the right or top side (see Details).
axlab	Where to put axis tick labels and multipliers. See Details.
normalize	Whether to scale each row of values to a maximum width of 1.
shownorm	Whether to display the normalization multipliers.
...	additional arguments passed to 'plot'.

Details

'kiteChart' displays each row of 'x' as a sequence of widths, allowing the relationships between those values and the dimension along which they occur (usually time) to be illustrated.

The values in x are scaled to a maximum polygon width of 1 if 'normalize' is TRUE. This is to avoid overlapping of the polygons. There may be some cases where the values can be displayed directly. If normalized, the multipliers will be displayed for each row on the right or top side of the chart unless 'shownorm' is FALSE. Remember to specify the 'mar' argument if more space at the top is needed.

The 'axlab' argument allows the user to place the axis tick labels and normalization multipliers on different axes. The default places the tick labels on the bottom and left sides of the plot and the multipliers on the right or top. Using 'axlab=c(3,4,1,2)' places the tick labels on the top and right and the multipliers on the left or bottom. The 'mar' argument may have to be adjusted.

The user can display raw values by default, or by setting 'varpos' to TRUE. Setting 'varpos' to a vector of positions will place the "kite lines" on those values. If there are no row names and the 'varlabels' argument is NA, the values of 'varpos' will be used as labels for each "kite line". The maximum extent of each "kite line" can be displayed by setting 'varscale' to TRUE. If 'varscale' is TRUE, one extra line will be added to the top margin. If 'varpos[1]' is not NA, 'normalize' is FALSE by default.

Value

The values of 'mar' that were current when 'kiteChart' was called.

Author(s)

Jim Lemon (Thanks to Michael Bedward for suggestions on the arguments and Nikolaus Lampadariou for the suggestions on displaying raw values)

See Also

polygon

Examples

```
testmat<-matrix(c(runif(50),sample(1:50,50),rnorm(50)+5,
 sin(1:50)),ncol=50,byrow=TRUE)
kiteChart(testmat,varlabels=c("Uniform","Sample","Normal","Sine"),
 timepos=seq(1,50,by=5),timex=FALSE)
# not enough space for the last label, add it
mtext("Sine",at=65,side=1,line=2)
# now show it with kite line maxima
kiteChart(testmat,varlabels=c("Uniform","Sample","Normal","Sine"),
 timepos=seq(1,50,by=5),timex=FALSE,varscale=TRUE)
mtext("Sine",at=65,side=1,line=2)
musicmat<-matrix(c(c(0.5,0.4,0.3,0.25,0.2,0.15,0.1,rep(0.05,44))+runif(51,0,0.05),
 c(0.1,0.2,0.3,0.35,0.4,0.5,0.4,rep(0.5,14),rep(0.4,15),rep(0.3,15))+runif(51,0,0.1),
 rep(0.15,51)+runif(51,0,0.1),
 c(rep(0,29),c(0.1,0.2,0.4,0.5,0.3,0.2,rep(0.05,16))+runif(22,0,0.05)),
 c(rep(0,38),c(rep(0.05,6),0.08,0.15,0.20,0.25,0.2,0.25,0.3)+runif(13,0,0.05))),
 ncol=51,byrow=TRUE)
kiteChart(musicmat,varlabels=c("Swing","Rock","Jazz","Disco","Rap"),
 main="An utterly imaginary chart of music popularity",
 timepos=seq(1,51,by=10),timelabels=seq(1950,2000,by=10),mar=c(5,4,4,2))
# now flip it to vertical, normalize and show the normalization factors
kiteChart(musicmat,varlabels=c("Swing","Rock","Jazz","Disco","Rap"),
 main="An utterly imaginary chart of music popularity",xlab="Style",
 timepos=seq(1,51,by=10),timelabels=seq(1950,2000,by=10),mar=c(5,4,4,2),
 timex=FALSE,normalize=TRUE,shownorm=TRUE)
```

---

l2010 *World lightning strike data from 2010*

Description

A list of two 50x100 matrices containing most of the world lightning strike data from 2010. It was produced by 'makeDensityMatrix' from 171 file (about 3 Gb) of data consisting in two geographic coordinates for the approximate location of each recorded strike and an estimated intensity of the strike in kVA.

Usage

```
data(l2010)
```

labbePlot *Display a L'Abbe plot*

Description

Display the percentages of successes for two conditions to be compared as circles, the area of which is proportional to the number of observations.

Usage

```
labbePlot(x,main="L'Abbe plot",xlab="Percent positive response with placebo",
ylab="Percent positive response with treatment",labels=NULL,col=NA,
circle.mag=0.5,add=FALSE,...)
```

Arguments

x	A list of either 2x2 tables or three element vectors (see Details).
main	The title of the plot.
xlab,ylab	The x and y axis labels as in 'plot'.
labels	Text strings that will be displayed in the center of the circles.
col	A list of colors for the circles.
circle.mag	A fudge factor for very small or very large numbers of observations.
add	Whether to add the information in 'x' to an existing L'Abbe plot.
...	additional arguments passed to 'plot'.

Details

The elements of 'x' may be tables in which rows represent the conditions being compared, with the comparison condition first (often "placebo") and the condition of interest (often "intervention") second. The columns represent the counts of successes and failures. The elements of 'x' can also be vectors with three numeric values, first the percentage of successes for the comparison condition, second the percentage of successes for the condition of interest and finally the number of observations. Tables and vectors can be mixed.

The radius of each circle is the square root of the number of observations multiplied by 'circle.mag'. This allows very small numbers of observations to be expanded and very large numbers to be reduced in size. As the area of each circle is proportional to the number of observations, 'circle.mag' must be the same for all circles. The user may wish to expand or contract all the circles on a plot so that they will fit within the box.

The labels, if not NULL, are displayed on the circles. The function tries to work out whether white or black text will be more easily read based on the background color and displays the text accordingly.

Value

nil

Author(s)

Jim Lemon - thanks to Whitney Melroy for asking for it.

See Also

draw.circle

Examples

```
# first fake something like the data from a clinical trial
didf<-data.frame(subject=1:50,interv=rep(c("therapist","ex-drinker"),each=25),
 outcome=sample(c("more","less"),50,TRUE))
# make it into a table
didf.tab<-table(didf$interv,didf$outcome)
# now mix in some raw percentages just for the example
didf2<-c(74,46,200)
didf3<-c(33,87,500)
x<-list(didf.tab,didf2,didf3)
labbecol<-list("red","green","blue")
labbePlot(x,main="Ex-drinkers vs therapists",
 xlab="Percent reduced drinking (ex-drinkers)",
 ylab="Percent reduced drinking (therapists)",
labels=list("A","B52","X117"),col=labbecol)
labbePlot(list(c(20,40,20)),col=list("purple"),labels=list("Z"),add=TRUE)
```

---

ladderplot *Ladder Plot*

Description

Makes a ladder plot, similar to parcoord but with more flexibility and graphical options.

Usage

```
ladderplot(x, ...)
## Default S3 method:
ladderplot(x, scale=FALSE, col=1, pch=19, lty=1,
xlim=c(0.5, ncol(x) + 0.5), ylim=range(x), vertical = TRUE, ordered=FALSE,...)
```

Arguments

x	A matrix or data frame with at least 2 columns.
scale	Logical, if the original data columns should be scaled to the unit (0-1) interval.
col	Color values to use for rows of 'x'. If longer than 1, its value is recycled.
pch	Point type to use. If longer than 1, its value is recycled.
lty	Line type to use. If longer than 1, its value is recycled.
xlim, ylim	Limits for axes.

vertical	Logical, if the orientation of the ladderplot should be vertical or horizontal.
ordered	Logical, if the columns in 'x' should be ordered.
...	Other arguments passed to the function stripchart.

Details

The function uses stripchart to plot 1-D scatter plots for each column in 'x'. Then points are joined by lines for each rows of 'x'.

Value

Makes a plot as a side effect. Returns 'NULL' invisibly.

Author(s)

Peter Solymos <solymos@ualberta.ca>

See Also

lines, points, stripchart

Almost identical function: parcoord

Examples

```
x<-data.frame(A=c(1:10), B=c(2:11)+rnorm(10))
y<-data.frame(x, C=c(1:10)+rnorm(10))
opar <- par(mfrow=c(1,3))
ladderplot(x)
ladderplot(x, col=1:10, vertical=FALSE)
ladderplot(y, col=1:10)
par(opar)

## examples from parcoord
## Not run:
if (require(MASS)) {
opar <- par(mfrow=c(2,3))
z1 <- state.x77[, c(7, 4, 6, 2, 5, 3)]
parcoord(z1, main="parcoord state.x77")
ladderplot(z1, pch=NA, scale=TRUE, main="ladderplot state.x77 original")
ladderplot(z1, main="ladderplot state.x77 original")
ir <- rbind(iris3[,,1], iris3[,,2], iris3[,,3])
z2 <- log(ir)[, c(3, 4, 2, 1)]
parcoord(z2, col = 1 + (0:149))
ladderplot(z2, scale=TRUE, col = 1 + (0:149),
    main="ladderplot iris original")
ladderplot(z2, col = 1 + (0:149))
par(opar)
}

## End(Not run)
```

legendg *Legend with grouped bars, lines or symbols*

Description

Displays a legend with more than one rectangle, symbol or line.

Usage

```
legendg(x,y=NULL,legend,fill=NULL,col=par("col"),
border=list("black"),lty,lwd,pch=NULL,angle=45,density=NULL,
bty="o",bg=par("bg"),box.lwd=par("lwd"),box.lty=par("lty"),
box.col=par("fg"),pt.bg=NA,cex=1,pt.cex=cex,pt.lwd=lwd,
pt.space=1,xjust=0,yjust=1,x.intersp=1,y.intersp=1,
adj=c(0,0.5),text.width=NULL,text.col=par("col"),merge=FALSE,
trace=FALSE,plot=TRUE,ncol=1,horiz=FALSE,title=NULL,
inset=0,xpd,title.col=text.col)
```

Arguments

x,y	Position of the legend as in 'legend'.
legend	Labels for the legend as in 'legend'.
fill	List of fill colors for the rectangles.
col	Color(s), perhaps as a list, for the symbols.
border	Border color(s) for the rectangles.
lty	Line type, currently ignored and set to 1.
lwd	Line width, currently ignored.
pch	List of symbols for the legend.
angle,density	Currently ignored.
bty	Legend box type to be displayed.
bg	Background color for the legend.
box.lwd,box.lty,box.col	Line width, type and color for the surrounding box.
cex	Character expansion for text.
pt.bg,pt.cex,pt.lwd	Background color, character expansion and line width for the symbols.
pt.space	Spacing for the symbols as a multiplier for 'strwidth("O")'.
xjust,yjust	Justification for the legend.
x.intersp,y.intersp	x and y character spacing for the legend text.
adj	Text adjustment.

`text.width,text.col`
: Width and color of the legend text.

`merge`
: Whether to merge points and lines.

`trace`
: Show how the legend is calculated.

`plot`
: Whether to plot the legend.

`ncol`
: Number of columns in the legend.

`horiz`
: Whether to display a horizontal legend.

`title`
: Title for the legend.

`inset`
: Inset distances for use with keywords.

`xpd`
: An optional value for '`par(xpd=)`'.

`title.col`
: Color for the legend title.

Details

'`legendg`' calls '`legend`' to display a legend with a blank space to the left of the labels. It then attempts to display groups of colored rectangles or symbols in that space depending upon the contents of either '`fill`' or '`pch`'. These should be in the form of a list with the number of elements equal to the number of labels, and one or more fills or symbols for each label. '`legendg`' will display up to four fills or symbols next to each label, allowing the user to label a group of these rather than just one per label.

Value

The value returned by '`legend`' returned invisibly.

Author(s)

Jim Lemon

See Also

legend

Examples

```
plot(0.5,0.5,xlim=c(0,1),ylim=c(0,1),type="n",
 main="Test of grouped legend function")
legendg(0.5,0.8,c("one","two","three"),pch=list(1,2:3,4:6),
 col=list(1,2:3,4:6),pt.space=1.5)
legendg(0.5,0.5,c("one","two","three"),fill=list(1,2:3,4:6))
# fake a line/point with text points
legendg(0.2,0.25,c("letter","number"),
 pch=list(c("-","A","-"),c("-","1","-")),
 col=list(rep(2,3),rep(3,3)))
```

`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)
```

`makeDensityMatrix` *Compute a matrix of counts from a list of x,y positions*

Description

Compute a matrix in which the counts in each cell represent the number of occurrences of that cell's coordinates in a list of x,y cooordinate values, optionally computing a second matrix of the average of the values attached to the coordinate observations.

Usage

```
makeDensityMatrix(x,y,z=NULL,nx=100,ny=50,zfun=c("mean","sum"),
xlim=c(-180,180),ylim=c(-90,90),geocoord=TRUE)
```

Arguments

`x,y`	Vectors of x and y coordinates. These are usually combined in a matrix or data frame of two columns.
`z`	Optional values attached to each coordinate pair. If these are present, it can be in a matrix or data frame of three columns, x, y and z.
`nx`	The number of "x" cells in the output matrix.
`ny`	The number of "y" cells in the output matrix.
`zfun`	The function to apply to the summed values attached to each coordinate pair. Currently defaults to mean, otherwise the sum is returned.
`xlim`	The extreme coordinates in the horizontal direction (see Details).
`ylim`	The extreme coordinates in the vertical direction (see Details).
`geocoord`	Whether to correct the matrix values for the areal distortion of the Mercator projection.

Details

'`makeDensityMatrix`' expects two vectors or a matrix or data frame with at least two columns. The function was written for geographic coordinates, but will also work for other numeric coordinates. An optional third vector or column of values for each coordinate will be processed.

Each coordinate pair adds to the count in that cell of the matrix. If there is a third element, that value is added to a second matrix in the same position. By default, the function computes the mean of all values in each cell. If '`zfun="sum"`', the sum of values in each cell will be returned.

As geograhic data sets may be very large, leading to memory problems, '`makeDensityMatrix`' can be run on small sections of the data set and the resulting matrices added together as long as the coordinate limits are consistent throughout.

Value

Either a matrix of counts of coordinate pairs within each cell or a list of two such matrices, the second containing the mean or sum of values associated with coordinate pairs.

Author(s)

Jim Lemon

See Also

densityGrid

Examples

```
x<-sample(1:20,400,TRUE)
y<-sample(1:20,400,TRUE)
z<-runif(400,5,20)
xyz<-makeDensityMatrix(x,y,z,nx=20,ny=20,xlim=c(1,10),ylim=c(1,10),
 geocoord=FALSE)
par(mar=c(7,3,2,3))
plot(0,xlim=c(1,10),ylim=c(1,10),type="n",xlab="",axes=FALSE)
box()
densityGrid(xyz,range.cex=c(1,4),xlim=c(1,10),ylim=c(1,10),
 red=c(0,0.5,0.8,1),green=c(1,0.8,0.5,0),blue=0,pch=15)
color.legend(3,-0.7,7,-0.2,c(5,10,15,20),
 rect.col=color.scale(1:4,cs1=c(0,0.5,0.8,1),cs2=c(1,0.8,0.5,0),cs3=0,alpha=1))
par(xpd=TRUE)
text(5,0.3,"Intensity")
points(c(3.5,4.5,5.5,6.5),rep(-1.7,4),pch=15,cex=1:4)
text(c(3.5,4.5,5.5,6.5),rep(-1.3,4),1:4)
text(5,-1,"Density")
par(xpd=FALSE)
```

---

makeIntersectList *Count set intersections*

Description

Create a list of set intersections from a matrix of indicators

Usage

```
makeIntersectList(x,xnames=NULL,sep="+")
```

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). 'x' can also be a matrix or data frame in which the first column contains object identifiers and the second contains attribute codes.
xnames	Optional user-supplied names for the attributes of x.
sep	A character to use as a separator for attribute labels.

Details

'makeIntersectList' reads a matrix (or data frame where all values are the same type) containing dichotomous values (either 0/1 or FALSE/TRUE) or labels (see next paragraph). In the first type of input, 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 'x' is all zeros, it will not be counted, but the second last element of the list returned contains the count of rows in 'x' and thus non-members can be calculated.

If a matrix (or data frame where all values are the same type) containing values other than 0/1 or TRUE/FALSE, it will be passed to 'categoryReshape' for conversion to a data frame as described above. See 'categoryReshape' for details of this.

makeIntersectList combines the set or attribute names to form intersection names. For the intersection of sets A and B, the name will be A+B (unless 'sep' is changed) 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, the total number of objects and the attribute codes.

Author(s)

Jim Lemon

See Also

intersectDiagram, pasteCols,categoryReshape

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)))
makeIntersectList(setdf)
ns<-sample(1:8,20,TRUE)
objects<-0
for(i in 1:length(ns)) objects<-c(objects,rep(i,ns[i]))
attributes<-"Z"
for(i in 1:length(ns)) attributes<-c(attributes,sample(LETTERS[1:8],ns[i]))
setdf2<-data.frame(objects[-1],attributes[-1])
makeIntersectList(setdf2)
```

maxEmptyRect *Find an empty space on a plot*

Description

Try to find the largest empty rectangle on a plot.

Usage

```
maxEmptyRect(ax,ay,x,y)
```

Arguments

ax,ay	The rectangle within which all of the points are contained. Usually the limits of a plot.
x,y	x and y positions of the points.

Details

'maxEmptyRect' searches the pairs of points on the plot to find the largest rectangular space within which none of the points lie. It does not guarantee that the space will be large enough to fit a legend or text.

Two alternatives are the 'largest.empty' function in the **Hmisc** package and the 'emptyspace' function. 'maxEmptyRect' appears to outperform 'emptyspace', particularly in running time. However, 'emptyspace' will sometimes find a "squarer" rectangle when 'maxEmptyRect' finds a slightly larger narrow rectangle.

Value

A list containing the area of the rectangle and the coordinates of the lower left and upper right corners (as used in 'rect') of the rectangle found.

Author(s)

Hans Borchers

References

A. Naamad, D. T. Lee, and W.-L. Hsu (1984). On the Maximum Empty Rectangle Problem. Discrete Applied Mathematics, 8: 267-277.

Examples

```
x<-runif(100)
y<-runif(100)
plot(x,y,main="Find the maximum empty rectangle",xlab="X",ylab="Y")
mer<-maxEmptyRect(c(0,1),c(0,1),x,y)
rect(mer$rect[1],mer$rect[2],mer$rect[3],mer$rect[4],border="red")
```

mtext3d *Display text in the margins of a 3D plot*

Description

Display text in the margins of a 3D plot.

Usage

```
mtext3d(edge,pmat,labels=TRUE,at=NULL,dist=0.3,xpd=NA,...)
```

Arguments

edge	which axis to calculate.
pmat	matrix to transform coordinates.
labels	labels to display in the margin.
at	position on the axis.
dist	Offset of the axis.
xpd	set clipping for display.
...	additional arguments passed to ptext3d.

Value

nil

Author(s)

Ben Bolker

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

A list including the return value for the first call to 'hist' (itself a list) and the values for the bar heights.

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

```
set.seed(1234)
l <- list(runif(10)*10,1:10,c(1,1,1,1,4,8))
multhist(l)
```

---

multivari *Function to draw a multivari chart*

Description

A multivari chart of one quantitative response variable depending on two to four categorical variables can be drawn.

Usage

```
multivari(var, fac1, fac2, fac3 = NULL, fac4 = NULL, data, sort = FALSE,
 fun = mean, pch = list(15, 16, 17), col = list("black", "blue", "red"),
 col0 = "black", cex = 1, fac.cex = 2, xlab.depth = 3, legend = FALSE,
 main = paste("multivari chart of", var), add = FALSE, ...)
```

Arguments

var	variable name (character string) or column index of response variable, required
fac1	variable name (character string) or column index of first level factor, required; precedes fac2 and fac3 (if present) in the hierarchy (see Details)
fac2	variable name (character string) or column index of second level factor, required; follows fac1 and precedes fac3 (if present) in the hierarchy (see Details)
fac3	variable name (character string) or column index of third level factor, optional; if present, fac3 is the last factor in the hierarchy (see Details)
fac4	variable name (character string) or column index of fourth level factor, optional; can only be specified if there is also a third level factor; if present, this factor is the first in the hierarchy (see Details), and separate multivari charts for the first three factors are drawn for each level of this factor
data	a data frame, required
sort	logical, specifying whether or not levels are sorted, when converting character vectors to factors (a single choice for all factors is needed); default: FALSE
fun	a function to be used in aggregation; default: mean
pch	a list of length 2 or 3, depending on whether or not fac3 is specified; the ith list element can be an individual plotting symbol (like the usual pch entry) or a vector of plot symbols for each level of fac_i
col	a list of length 2 or 3, depending on whether or not fac3 is specified; the ith list element can be an individual color or a vector of colors for each level of fac_i; this color specification is used for the plot symbols of fac_i levels and for the lines connecting the symbols for the next level in the hierarchy
col0	the color for the first line to be drawn
cex	the size of axis annotation text (annotation of the fourth level header is 1.5 times this size)
fac.cex	a multiplier for cex; plot symbol sizes are fac.cex*cex; default: 2
xlab.depth	labels for the horizontal axis are printed down to this level of the hierarchy (default: 3); if the depth is reduced, different plot symbols should be used, and a legend should be drawn
legend	logical determining whether or not a legend should be drawn (default: FALSE); the function determines wether top right or bottom right yields a better position (it is not guaranteed that there is no overlab); if this does not work well, one can manually draw a legend in the outer margin
main	title, as usual; a default is provided

add	logical; add to an existing plot (which of course has to have suitable axis limits)?; default: FALSE; note that horizontal axis labeling will always be printed by function multivari, while vertical axis labeling will be omitted for add=TRUE
...	further arguments to functions plot, lines, points, mtext

Details

The function is inspired by Minitabs behavior for multivari charts (see also Bruno Scibilia's blog which is linked in the references). It does not attempt to visualize individual observations.

A multivari chart mainly serves exploratory purposes. It works particularly well with balanced data, but can also be used for messy data. multivari can visualize the dependence of a single quantitative variable on up to four factors (i.e., interactions of order up to four can be visualized). The display is hierarchical: for factors later in the hierarchy, conditional means given level combinations of factors earlier in the hierarchy are displayed. Therefore, the order of the factors can make a big difference in the display. If there is no natural order, it may be worthwhile to inspect several orders.

For interactions with two factors only, it is often preferrable to use function interaction.plot or raw.means.plot.

Value

a list of (lists of) data frames with summary statistics to be plotted

Author(s)

Ulrike Groemping

References

Scibilia, Bruno (2013). Using Multi-Vari Charts to Analyze Families of Variations. https://blog.minitab.com/en/using-variability-charts-to-analyze-call-center-wait-times.

See Also

See also interaction.plot, raw.means.plot

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
## Not run:
require(car)
multivari("cycles", "len", "load", "amp", data=Wool,
    col=list("black","red",c("grey70","grey45","grey20")),
    pch=list(15,17,8), legend=TRUE, xlab.depth = 2, lwd=2)
multivari("cycles", "load", "len", "amp", data=Wool,
    col=list("black",c("red","blue","darkgreen"),
    c("grey70","grey45","grey20")),
    pch=list(15,17,8), legend=TRUE, xlab.depth = 2, lwd=2)
```

```
## create a fake fourth factor
fakedat <- rbind(cbind(newfac="blabla",Wool),cbind(newfac="albalb",Wool))
## make it character for demonstrating the effect of sort option
fakedat$newfac <- as.character(fakedat$newfac)

## default: sort order in the data is respected (order of unique is used)
multivari("cycles", "load", "len", "amp", "newfac", data=fakedat,
    col=list("black",c("red","blue","darkgreen"),
    c("grey70","grey45","grey20")),
    pch=list(15,17,8), legend=TRUE, xlab.depth = 2, lwd=2, cex=0.8)

## sort=TRUE: levels are sorted (order of sort(unique))
multivari("cycles", "load", "len", "amp", "newfac", data=fakedat,
    col=list("black",c("red","blue","darkgreen"),
    c("grey70","grey45","grey20")),
    pch=list(15,17,8), legend=TRUE, xlab.depth = 2, lwd=2, cex=0.8,
    sort=TRUE)

## End(Not run)
```

`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 symbolsn
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,maxpct=20,wrmar=c(4,5,6,5),scale.factor=30,
 speed.col=c("#dab286","#fe9a66","#ce6733","#986434"),
 speed.width=NA,show.legend=TRUE,legend.pos=NA,...)
```

Arguments

`windagg`	A matrix of percentages with the rows representing speed ranges and the columns indicating wind directions.
`maxpct`	The maximum percentage displayed on the radial grid.
`wrmar`	Plot margins for the diagram.
`scale.factor`	The scale factor for the diagram.
`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 and Alejo for finding the problem with heavily prevailing winds.)

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(maxpct=20,scale.factor=30,
 speed.col=c("#dab286","#fe9a66","#ce6733","#986434"),
 speed.width=NA,legend.pos=NA)
```

Arguments

maxpct	The maximum percentage to display on the radial grid.
scale.factor	The scale factor for the plot.
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)
```

---

p2p_arrows *Draw arrows between points*

Description

Displays arrows on an existing plot between specified points.

Usage

```
p2p_arrows(x1,y1,x2,y2,space=0.05,col=par("fg"),...)
```

Arguments

x1	Starting x positions for the labels.
y1	Starting y positions for the labels.
x2	Ending x positions for the labels.
y2	Ending y positions for the labels.
space	The proportion of the distance between the points to leave as space before and after the arrow.
col	Color(s) for the arrows.
...	Extra arguments passed to 'arrows'.

Details

'p2p_arrows' displays arrows on a plot between one or more pairs of specified points.

Value

nil

Author(s)

Jim Lemon

See Also

arrows

`panes` *Prepare a "panel" type layout*

Description

Split the graphics device into a "panel" type layout for a group of plots

Usage

```
panes(mat=NULL,widths=rep(1,ncol(mat)),heights=rep(1,nrow(mat)),
 nrow=2,ncol=2,mar=c(0,0,1.6,0),oma=c(2.5,1,1,1))
```

Arguments

`mat`	A matrix representing the number of panes to be created and their order of plotting.
`widths,heights`	The widths and heights of the panes. See 'layout'.
`nrow,ncol`	The numbers of rows and columns in the layout. See 'par(mfrow)'.
`mar`	The margins for each plot in the panes.
`oma`	The outer margins for the entire group of panes.

Details

'panes' combines the information for displaying a set of plots in a "panel" layout. The default values will usually produce the desired result by calling 'par(mfrow)'. If 'mat' is not NULL, the 'layout' function will be called instead of 'par(mfrow)'. The two methods are included for the convenience of the user.

Note that 'panes' does not produce any plots and that the user must call 'tab.title' to get the "look" of the panel plot. The overall title is usually centered at the left edge (as in the example) or in the center of one of the plots in the bottom row.

Value

The values of 'par' options that existed when 'panes' was called. This list is usually used to restore those values.

Author(s)

Jim Lemon

See Also

par,layout

Examples

```
y<-runif(8)
oldpar<-panes(matrix(1:4,nrow=2,byrow=TRUE))
par(mar=c(0,2,1.6,0))
boxplot(y,axes=FALSE)
axis(2)
box()
par(mar=c(0,0,1.6,2))
tab.title("Boxplot of y",tab.col="#88dd88")
barplot(y,axes=FALSE,col=2:9)
axis(4)
box()
tab.title("Barplot of y",tab.col="#88dd88")
par(mar=c(2,2,1.6,0))
pie(y,col=2:9)
tab.title("Pie chart of y",tab.col="#88dd88")
box()
par(mar=c(2,0,1.6,2))
plot(y,xaxs="i",xlim=c(0,9),axes=FALSE,col=2:9)
axis(4)
box()
tab.title("Scatterplot of y",tab.col="#88dd88")
# center the title at the left edge of the last plot
mtext("Test of panes function",at=0,side=1,line=0.8,cex=1.5)
panes(matrix(1:3,ncol=1),heights=c(0.7,0.8,1))
par(mar=c(0,2,2,2))
plot(sort(runif(7)),type="l",axes=FALSE)
axis(2,at=seq(0.1,0.9,by=0.2))
box()
tab.title("Rising expectations",tab.col="#ee6666")
barplot(rev(sort(runif(7))),col="blue",axes=FALSE)
axis(2,at=seq(0.1,0.9,by=0.2))
box()
tab.title("Diminishing returns",tab.col="#6666ee")
par(mar=c(4,2,2,2))
tso<-c(0.2,0.3,0.5,0.4,0.6,0.8,0.1)
plot(tso,type="n",axes=FALSE,xlab="")
# the following needs a Unicode locale to work
points(1:7,tso,pch=c(rep(-0x263a,6),-0x2639),cex=2)
axis(1,at=1:7,
 labels=c("Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday","Monday"))
axis(2,at=seq(0.1,0.9,by=0.2))
box()
tab.title("The sad outcome",tab.col="#66ee66")
mtext("A lot of malarkey",side=1,line=2.5)
par(oldpar)
```

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="+")
```

paxis3d *Display text in the margins of a 3D plot*

Description

Display text in the margins of a 3D plot.

Usage

```
paxis3d(edge,pmat,at=NULL,labels=TRUE,tick=TRUE,
 pos=NULL,nticks=5,ticklen=0.05,labdist=0.15,xpd=NA,...)
```

Arguments

edge	which axis to calculate.
pmat	matrix to transform coordinates.
at	position on the axis.
labels	labels to display in the margin.
tick	whether to draw axis tick marks.
pos	axis position relative to other axes.
nticks	number of tick marks.
ticklen	length of tick marks as a proportion of plot dimensions.
labdist	distance of labels from axis.
xpd	parameter to set plot clipping.
...	additional arguments passed to ptext3d.

Value

nil

Author(s)

Ben Bolker

Examples

```
x <- 1:10
y <- 1:10
z <- outer(x,y,function(x,y) { 3*sin(2*pi*x)/(2*pi*x)+exp(y/10)+(x*y)/1000 })
par(mar=c(5,10,2,2))
pp <- perspx(x,y,z,ticktype="detailed",phi=30,theta=80,nticks=3,r=10,
     axes=FALSE)
## axis labels not drawn when axes=FALSE
paxis3d("X-",pp,at=c(1,2,9))
paxis3d("Y+",pp)
```

```
paxis3d("Z-",pp)
mtext3d("X-",pp,expression(alpha^sqrt(beta)))
## if you want labels parallel to axis, still have to figure out 'srt'
##   by trial and error
mtext3d("Y+",pp,expression("velocity ("*gamma*", furlongs/fortnight)"),
       xpd=NA,srt=6)
mtext3d("Z-",pp,"Range\n(r*)",dist=0.5)
```

---

perspx *Display perspective plot*

Description

Display an enhanced perspective plot with additional return values

Usage

```
perspx(x,y,z,...)
```

Arguments

x,y,z	x, y and z coordinates to plot.
...	Other arguments passed to 'persp'.

Details

Displays 'z' values plotted on an x,y grid.

Value

A list with three elements, the ranges of 'x', 'y' and 'z'.

Author(s)

Ben Bolker

Examples

```
x <- 1:10
y <- 1:10
z <- outer(x,y,function(x,y) { 3*sin(2*pi*x)/(2*pi*x)+exp(y/10)+(x*y)/1000 })
par(mar=c(5,10,2,2))
pp <- perspx(x,y,z,ticktype="detailed",phi=30,theta=80,nticks=3,r=10,
 axes=FALSE)
```

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

Description

Places labels on a pie chart

Usage

```
pie.labels(x=0,y=0,angles,labels,radius=1.05,bg="white",border=TRUE,
 minangle=NA,boxed=FALSE,explode=0,...)
```

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 1.05.
`bg`	The color of the rectangles on which the labels are displayed.
`border`	Whether to draw borders around the rectangles.
`minangle`	Minimum angle between labels.
`boxed`	Whether to use '`text`' or '`boxed.labels`' to display the labels.
`explode`	How much the pie chart has been "exploded".
`...`	Arguments passed to '`text`' or '`boxed.labels`'.

Details

Labels may be placed within the pie (radius less than the pie radius), on the edge or outside as in the examples below. If within the pie, it is probably best to use '`boxed=TRUE`'.

If some labels overlap, passing a value in radians for '`minangle`' may be used to spread them out.

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, spreadout

Examples

```
pieval<-c(2,1,3,94)
plot(0,xlim=c(1.5,5),ylim=c(1,5),type="n",axes=FALSE,xlab="",ylab="")
box()
bisect.angles<-floating.pie(3,3,pieval,explode=c(0.1,0.2,0.3,0))
pie.labels(3,3,bisect.angles,c("two","one","three","ninety\nfour"),
 minangle=0.2,,explode=c(0.1,0.2,0.3,0))
```

---

pie3D *Display a 3D pie chart*

Description

Displays a 3D pie chart with optional labels.

Usage

```
pie3D(x,edges=NA,radius=1,height=0.1,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,mar=c(4,4,4,4),pty="s",...)
```

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 (see examples)
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.
mar	Margins around the pie.
pty	Whether to force a square plot region or not. (see Details)
...	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 these outside 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 or change the position as in the example below.

In order to make the dimensions of the pie reasonably accurate, a square plot region ('pty="s"') is the default. If 'pty' is set to "m", the user can change the margins, usually resulting in a non-square plot area. This will probably distort the pie somewhat.

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. As a last resort, the user can try setting 'explode' to zero. This only draws the top and outer sides of each sector.

Also due to the sector drawing method, setting 'theta' to values smaller than about pi/8 or larger than about pi/4 will produce obviously misaligned sectors.

Contributed fixes and improvements: thanks to Jesse Brown for the "shade" fix and Qinghua Zhao for alerting me to the problem with labels and margins

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")
# grab the radial positions of the labels
lp<-pie3D(pieval,radius=0.9,labels=pielabels,explode=0.1,main="3D PIE OPINIONS")
# lengthen the last label and move it to the left
pielabels[4]<-"We cannot survive without our pies"
lp[4]<-4.8
# specify some new colors
pie3D(pieval,radius=0.9,labels=pielabels,explode=0.1,main="3D PIE OPINIONS",
```

```
col=c("brown","#ddaa00","pink","#dd00dd"),labelpos=lp)
```

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

Description

Displays labels on a 3D pie chart.

Usage

```
pie3D.labels(radialpos,radius=1,height=0.1,theta=pi/6,
 labels,labelcol=par("fg"),labelcex=1.5,labelrad=1.25,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
`labelrad`	The expansion for the labels around the pie.
`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)
```

---

placeLabels *Place labels in boxes*

Description

Places labels in boxes on an existing plot

Usage

```
placeLabels(x,y=NA,labels,pointer=TRUE,cex=1,labelcol=par("fg"),
labelbg="white",border=par("fg"),pointercol=par("fg"),
pch=1,col=1,bg="white",flagcol="red")
```

Arguments

x,y	x and y position of the centers of the labels. 'x' can be an xy.coords list.
labels	Text strings
pointer	Whether to draw a line segment from the label to the points labeled.
cex	Character expansion. See 'text'.
labelcol	The color(s) of the text in the labels.
labelbg	The background color(s) for the labels.
border	The color(s) for the borders around the rectangles.
pointercol	The color(s) of the pointer lines.
pch	The symbol(s) to use when redisplaying the original points (see Details).
col	The color(s) of the original points.
bg	The background color(s) of the original points.
flagcol	The color to use for "flagging" each point.

Details

'placeLabels' steps through the points indexed by 'x' and 'y', allowing the operator to manually place the labels for each point. Each point is "flagged" by displaying a small colored circle (red by default). When the label for that point has been placed, the original symbol is displayed and the next point is flagged.

Each point and label can have different colors and backgrounds.

Value

nil - adds labels to an existing plot.

Note

This function is handy for one-off plots with a moderate number of points. It can be very useful for plots with clumps of points.

Author(s)

Jim Lemon - thanks to Marna Wagley for the idea.

See Also

spread.labels, thigmophobe.labels

Examples

```
 # won't check because of the call to locator
 ## Not run:
  x<-rnorm(10)
  y<-rnorm(10)
  plot(x,y)
  placeLabels(x,y,LETTERS[1:10],flagcol="purple")

## End(Not run)
```

---

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(col)'
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,main="Basic plotCI")
plotCI(1:10,y,err,2*err,lwd=2,col="red",scol="blue",
 main="Add colors to the points and error bars")
err.x<-runif(10)
err.y<-runif(10)
```

```
plotCI(1:10,y,err.y,pt.bg=par("bg"),pch=21,xlim=c(0,11),
 main="plotCI with extra space on the x axis")
plotCI(1:10,y,err.x,pt.bg=par("bg"),pch=21,err="x",add=TRUE)
mtext("for adding horizontal error bars",3,0.5)
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,main="plotCI with labels at points")
text(1:10,y,labs)
```

---

plotH *Scatterplot with histogram-like bars.*

Description

Scatterplot with histogram-like bars; a modification of 'plot(...,type="h")'.

Usage

```
 plotH(x,...)

 ## S3 method for class 'formula'
plotH(x,data=NULL,xlab=names(mf)[2],ylab=names(mf)[1],...)

 ## Default S3 method:
plotH(x,y,xlab=paste(deparse(substitute(x))),
  ylab=paste(deparse(substitute(y))),width=0.6,ylim=NULL,col="gray",...)
```

Arguments

x	Vector of x-coordinates or a formula of the form y~x (see below for y).
y	Vector of y-coordinates.
xlab	A string for labeling the x-axis.
ylab	A string for labeling the y-axis.
data	The data frame from which the formula should be evaluated.
width	A numeric that indicates the width of the bars.
ylim	A vector of length two that indicates the limits over which to plot the y-axis. See details.
col	A string that indicates the fill color for the bars.
...	Additional arguments sent to the 'plot' or 'barplot' functions.

Details

'plotH' is meant to be a modification of the type="h" version of 'plot' such that the "bars" appears as actual rectangles rather than vertical lines. It defaults so that the lower bound of the y-axis is 0; change to 'ylim=NULL' to over-ride this default (and return to the default used in 'plot'.

A pass-through to 'barplot' is used if the 'x' (or "RHS") variable is categorical.

Value

None, but a plot is produced.

Note

This function is currently experimental.

Author(s)

Derek Ogle

See Also

plot, barplot

Examples

```
d<-data.frame(x=c(1,5,10:20),y=runif(13)+1,
              yn1=runif(13)-0.5,yn2=runif(13)-2,
              g=factor(sample(c("A","B","C"),13,replace=TRUE)))
# new plotH function with formula notation
plotH(y~x,data=d)
# old plot() function with formula notation -- for comparison's purpose
plot(y~x,data=d,type="h")
# new function over-riding default ylim, increasing bar width,
# and changing bar color
plotH(y~x,data=d,ylim=range(d$y),width=0.9,col="red")
# handling some negative values
plotH(yn1~x,data=d) # not so good, because of default ylim
plotH(yn1~x,data=d,ylim=c(0,max(d$yn1)))  # old look
# handling all negative values
plotH(yn2~x,data=d)
plotH(yn2~x,data=d,ylim=range(d$yn2))  # old look
# example of pass-through to barplot
smry<-by(d$y,d$g,mean)
plotH(levels(d$g),smry,ylab="Mean of Random Variable",xlab="Group")
# example of non-formula usage
x1 <- d$x
y1 <- d$y
plotH(x1,y1,col="blue")
```

plot_bg *Add a background color to a plot*

Description

Displays a colored rectangle over the entire area of a plot

Usage

```
plot_bg(col="lightgray")
```

Arguments

col	The color of the background

Details

'plot_bg' is probably only useful when part of the 'do.first' argument in another plot function to add a background color to the plot.

Value

nil

Author(s)

Jim Lemon

Examples

```
barp(1:5,do.first="plot_bg()",col=1:5)
```

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.

If 'add=TRUE' is passed as one of the additional arguments, the values will be added to the current plot. If a 'radial.lim' argument was passed on the initial plot, it must be passed again to add values or the values will be displayed incorrectly.

Usage

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

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.
loglen	Whether to log transform the 'length' values. Only base 10 logs are available.
explab	Whether to use the default fixed (FALSE) or exponential (TRUE) notation for the radial labels.
...	additional arguments passed to 'radial.plot' and then to 'plot'.

Value

A list of the parameters altered by radial.plot.

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)
oldpar<-polar.plot(testlen,testpos,main="Test Clockwise Polar Plot",
 radial.lim=c(0,15),start=90,clockwise=TRUE,lwd=3,line.col=4)
# reset everything
par(oldpar)
```

`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. A good starting point for the two colors is the color of the background and the RGB components of that color multiplied by 0.8. Use a smaller multiplier for a darker shadow.

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)
```

`print.brklist` *Display the output of brkdnNest*

Description

Displays the list of values produced by '`brkdnNest`'.

Usage

```
 ## S3 method for class 'brklist'
print(x,...)
```

Arguments

`x`	a list of summary values produced by '`\link{brkdnNest}`'
`...`	additional arguments passed to '`print`'.

Details

'`print.brklist`' displays frequency tables produced by '`brkdnNest`'. It is mainly for convenience, but does make a nicer display than when passed directly to '`print`'

Value

nil

Author(s)

Jim Lemon

See Also

brkdnNest

Examples

```
printbrktest<-data.frame(A=c(sample(1:10,99,TRUE),NA),
 B=sample(c("Yes","No"),100,TRUE),
 C=sample(LETTERS[1:3],100,TRUE))
pbt<-brkdnNest(A~B+C,printbrktest)
print(pbt)
```

`propbrk` *Calculate the proportion of specified values in a vector*

Description

Calculates the proportion of values in a vector that are equal to a specified value.

Usage

```
propbrk(x,trueval=TRUE,na.rm=TRUE)
```

Arguments

`x`	a character, factor or numeric vector.
`trueval`	the value to be matched in '`x`'.
`na.rm`	whether to remove NA values.

Details

'`propbrk`' calculates the proportion of values matching a specified value. It is mainly to allow proportions to be calculated in the '`brkdnNest`' function. It always discards NAs in '`x`' when summing the number equal to '`trueval`', but respects the '`na.rm`' argument when calculating the total number of values in '`x`'.

Value

nil

Author(s)

Jim Lemon

See Also

brkdnNest

Examples

```
propbrk(sample(LETTERS,100,TRUE),trueval="M")
```

---

psegments3d *Draw segments on a 3D plot*

Description

Draw segments on a 3D plot defined by a list of coordinates

Usage

```
psegments3d(x,y=NULL,z=NULL,pmat,...)
```

Arguments

x,y,z	x, y and z coordinates to plot. 'x' may be a list with three components.
pmat	matrix to transform coordinates.
...	Other arguments passed to 'segments'.

Details

Draws segments on a perspective plot.

Value

nil

Author(s)

Ben Bolker

---

ptext3d *Display text on a 3D plot*

Description

Display text on a 3D plot defined by a list of coordinates

Usage

```
ptext3d(x,y=NULL,z=NULL,texts,pmat,...)
```

Arguments

x,y,z	x, y and z coordinates to plot. 'x' may be a list with three components.
pmat	matrix to transform coordinates.
texts	text to display.
...	Other arguments passed to 'segments'.

Details

Draws text on a perspective plot.

Value

nil

Author(s)

Ben Bolker

---

pyramid.plot *Pyramid plot*

Description

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

Usage

```
pyramid.plot(lx,rx,labels=NA,top.labels=c("Male","Age","Female"),
 main="",laxlab=NULL,raxlab=NULL,unit="%",lxcol,rxcol,gap=1,space=0.2,
 ppmar=c(4,2,4,2),labelcex=1,add=FALSE,xlim,show.values=FALSE,ndig=1,
 do.first=NULL)
```

Arguments

lx,rx	Vectors or a matrix or data frame (see Details) which should be of equal length.
labels	Labels for the categories represented by each pair of bars. There should be a label for each lx or rx value, even if empty. If 'labels' is a matrix or data frame, the first two columns will be used for the left and right category labels respectively.
top.labels	The two categories represented on the left and right sides of the plot and a heading for the labels in 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.
lxcol,rxcol	Color(s) for the left and right sets of bars. Both of these default to 'rainbow(length(labels))'.
gap	One half of the space between the two sets of bars for the 'labels' in user units.
space	Space between the bars. Should be 0 <= space < 1.
ppmar	Margins for the plot (see Details).
labelcex	Expansion for the category labels.
add	Whether to add bars to an existing plot. Usually this involves overplotting a second set of bars, perhaps transparent.
xlim	Optional x limit for the plot (see Details).
show.values	Whether to display 'lx' and 'rx' at the ends of the bars.
ndig	The number of digits to round the values if displayed.
do.first	Optional expression to evaluate before displaying anything.

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 'labels' is a matrix or data frame of at least two columns, the first column will be displayed on the on the left side of the gap in the center, and the second on the right. This will almost always require increasing the gap width and perhaps also specifying a wider plotting device. Displaying the values will usually require increasing the left and/or right margins of the plot, or setting 'xlim' larger than the largest value.

If a gap width of zero is passed, the category labels will be displayed at the left and right extents of the plot. This usually requires setting 'xlim' to values larger than the maximum extent of 'lx' and 'rx'. The user can pass two different values to 'xlim', but this is almost always a bad idea, as the lengths of the bars will not be in the same proportion to the values on the left and right sides. Both the bars and category labels are vertically centered on integer values, allowing the user to easily add components to the plot.

'lx' and 'rx' are the values specifying the left and right extents of the left and right bars respectively. If both 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 or with a gap of zero. 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 (thanks to Susumu Tanimura for the patch that omits ticks for NA values in vector input and Igor Rebeiro for the space argument)

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+")
mcol<-color.gradient(c(0,0,0.5,1),c(0,0,0.5,1),c(1,1,0.5,1),18)
fcol<-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",lxcol=mcol,rxcol=fcol,
 gap=0.5,show.values=TRUE))
# 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)
# 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",lxcol=c("#ff0000","#eeee88","#0000ff"),
 rxcol=c("#ff0000","#eeee88","#0000ff"),laxlab=c(0,10,20,30),
 raxlab=c(0,10,20,30),top.labels=c("Males","Age","Females"),gap=4,
 do.first="plot_bg(\"#eedd55\")")
# put a box around it
box()
# give it a title
mtext("Porridge temperature by age and sex of bear",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")
```

---

radial.grid *Display a radial grid*

Description

'radial.grid' displays a radial grid for the 'radial.plot' and 'radial.pie' functions.

Usage

```
radial.grid(labels=NA,label.pos=NULL,radlab=FALSE,radial.lim=NULL,
 start=0,clockwise=FALSE,label.prop=1.1,grid.pos=seq(0.25,1,0.25),
 rad.col="gray",grid.col="gray",grid.bg="transparent",show.radial.grid=TRUE,
 start.plot=FALSE)
```

Arguments

labels	The labels to display around the circumference of the grid.
label.pos	Radial positions for the labels.
radlab	Whether to rotate the labels to a radial orientation.
radial.lim	Optional radial limits for the circular plot. If specified, these must be the same as the radial limits of the original plot.
start	The zero position on the plot in the units of 'label.pos'.
clockwise	Whether to increase angles clockwise rather than the default counterclockwise.
label.prop	Proportion of 'radial.lim' to place the labels.
grid.pos	Radial positions for the circular grid lines.
rad.col	Color for the radial grid lines.
grid.col	Color for the circumferential grid lines.
grid.bg	Background color for the radial grid.
show.radial.grid	Whether to display the radial lines on the grid.
start.plot	If TRUE, sets up a blank radial grid.

Value

nil

Author(s)

Jim Lemon

---

radial.pie *Plot sectors/annuli on a circular grid of 0 to 2*pi radians*

Description

Plot numeric values as sectors with optional annuli on a circular field in the directions defined by angles in radians.

Usage

```
radial.pie(radial.extents,sector.edges=NULL,
sector.colors=NULL,cs1=c(0,1),cs2=c(0,1),cs3=c(0,1),
alpha=1,labels=NA,label.pos=NULL,radlab=FALSE,start=0,
clockwise=FALSE,label.prop=1.1,radial.lim=NULL,main="",xlab="",ylab="",
mar=c(2,2,3,2),show.grid=TRUE,show.grid.labels=4,show.radial.grid=TRUE,
grid.col="gray",grid.bg="transparent",grid.unit=NULL,
radial.labels=NULL,boxed.radial=TRUE,add=FALSE,...)
```

Arguments

radial.extents	A numeric data vector or list. If 'radial.extents' is a list, the elements of the list will be considered separate data vectors.
sector.edges	A numeric vector of positions in radians. These are interpreted as beginning at the right (0 radians) and moving counterclockwise unless 'clockwise' is TRUE.
sector.colors	Optional colors for the sectors and annuli. Defaults to 'rainbow(nsectors)' with fading outward if annuli are specified.
cs1, cs2, cs3, alpha	Color scaling arguments - see color.scale.
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.
label.prop	The label position radius as a proportion of the maximum line length.
radial.lim	The inner and outer radial limits for the plot. Defaults to the range of radial.extents, although zero to 'max(radial.extents)' is often what is wanted.
main	The title for the plot.
xlab,ylab	Normally x and y axis labels are suppressed.
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	Whether and where to display labels for the grid - see Details.
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.unit	Optional unit description for the grid.
radial.labels	Optional labels for the radial grid. The default is the values of radial.lim.
boxed.radial	Whether to use boxed.labels or text for radial labels.
add	Whether to add one or more series to an existing plot.
...	Additional arguments are passed to 'plot'.

Details

'radial.pie' displays a plot of radial sectors with optional annular sections centered at the midpoint of the plot frame, the lengths corresponding to the numeric magnitudes of 'radial.extents'.

If more series are added to an existing plot, 'radial.pie' will try to maintain the current plot parameters. However, it seems unlikely that adding series would be sensible in 'radial.pie'. This argument may be dropped if it proves useless.

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. If the 'show.grid.labels' argument is a number from 1 to 4, the labels will be placed along a horizontal or vertical radius. The numbers represent the same positions as in 'axis', with the default (4) on the right. To suppress these labels, pass zero or FALSE.

'radial.pie' works somewhat differently from the 'radial.plot' family and is still under development. I have released it in order to get feedback to improve both the design and the programming. If successful, I hope to merge the code with the 'radial.plot' function.

Value

The 'par' values that are changed in the function as they were at the time 'radial.pie' was called.

Author(s)

Jim Lemon - thanks to Patrick Jemison for asking for it.

See Also

radial.plot

Examples

```
pie1<-c(3,6,5,4,7,8,9,1,4)
pie2<-list(0:3,1:6,2:5,1:4,0:7,4:8,2:9,0:1,0:4)
pie3<-sample(10:60,36)
pie4<-list(sort(sample(1:60,8)))
for(sector in 2:36) pie4[[sector]]<-sort(sample(1:60,8))
oldpar<-radial.pie(pie1,labels=LETTERS[1:9])
radial.pie(pie2,labels=letters[2:10])
radial.pie(pie3,labels=1:36)
radial.pie(pie4,labels=1:36)
# restore the par values
par(oldpar)
```

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=NA,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=4,show.radial.grid=TRUE,rad.col="gray",
grid.col="gray",grid.bg="transparent",grid.left=FALSE,grid.unit=NULL,
point.symbols=1,point.col=par("fg"),show.centroid=FALSE,radial.lim=NULL,
radial.labels=NULL,boxed.radial=TRUE,poly.col=NA,add=FALSE,
loglen=FALSE,explab=FALSE,...)
```

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 NULL, 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, (t)ext, 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	Whether and where to display labels for the grid - see Details.
show.radial.grid	Whether to draw radial lines to the plot labels.
rad.col	Color of the radial lines on the grid.
grid.col	Color of the circumferential lines on the 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) or if 'rp.type' is "t", the text that will be displayed.
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, or if loglen is TRUE, the corresponding log values.
boxed.radial	Whether to use boxed.labels or text for radial labels.
poly.col	Fill color if polygons are drawn. Use NA for no fill.
add	Whether to add one or more series to an existing plot.
loglen	Whether to log transform the 'length' values. Only base 10 logs are available. Keep in mind that the values actually plotted will be the logarithms, although the exponentiated logs are displayed.
explab	Whether to use the default fixed (FALSE) or exponential (TRUE) notation for the radial labels.
...	Additional arguments are passed to 'plot'.

Details

'radial.plot' displays a plot of radial lines, polygon(s), symbols, text 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. Note that if log transformation is requested with 'loglen', the values plotted will be the logs, not the values displayed on the plot. 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. Note that if the observations are not taken at equal intervals around the circle, the centroid may not mean much.

The 'text' option for 'rp.type' allows the user to place text at each point. It is useful for adding labels at arbitrary points on an existing plot or perhaps labelling points with letters or digits rather than different symbols. See the last example.

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.

If more series are added to an existing plot, 'radial.plot' will try to maintain the current plot parameters. Resetting the parameters after doing the initial plot will almost certainly mess up any series that are added. Series that are added will be plotted "on top" of the existing plot, possibly overplotting other things. If the added series have a larger range than the initial series, set 'radial.lim' to account for this in the initial plot, and if 'radial.lim' is specified in the initial plot, remember to repeat it for added series as in the example.

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. If the 'show.grid.labels' argument is a number from 1 to 4, the labels will be placed along a horizontal or vertical radius. The numbers represent the same positions as in 'axis', with the default (4) on the right.

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.

Value

The 'par' values that are changed in the function as they were at the time 'radial.plot' was called.

Note

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'

Thomas Steiner for the request for the 'radial.lim' and 'radial.labels' modifications

Evan Daugharty for requesting the 'add' argument

James MacCarthy for requesting better radial labels

Steve Ellison for noticing that the return values of the functions had changed

Don Dennerline for requesting the rank clock

Mehdi Nellen for the different colors for the radial and circumferential lines for the grid

Mayeul Kauffmann for noticing the radial label bug when a separate radial.grid was included

Ogbos Okike for requesting a text option for rp.type

Keziah Conroy for requesting the log option

Author(s)

Jim Lemon

See Also

polar.plot,clock24.plot

Examples

```
testlen<-runif(10,0,10)
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,rad.col="lightblue")
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",
 labels=LETTERS[1:8],label.pos=seq(0,14*pi/8,length.out=8))
# now do a 12 o'clock start with clockwise positive
radial.plot(testlen,testpos,start=pi/2,clockwise=TRUE,show.grid.labels=2,
 rp.type="s",main="Test Symbols (clockwise)",radial.lim=c(0,3.5),
 point.symbols=16,point.col="green",show.centroid=TRUE,
 labels=LETTERS[1:6],label.pos=seq(0,10*pi/6,length.out=6))
# 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",show.grid.labels=3)
# add the names of the ions to the plot
radial.plot(ions,rp.type="t",point.symbols=ion.names,radial.lim=c(0,5),
 add=TRUE)
# add points inside the polygon - radial.lim is supplied by plotrix_env
radial.plot(ions-0.4,rp.type="s",point.symbols=4,point.col="red",add=TRUE)
radmat<-matrix(c(sample(1:4,4),sample(1:4,4),sample(1:4,4),sample(1:4,4),
 sample(1:4,4),sample(1:4,4),sample(1:4,4),sample(1:4,4),
 sample(1:4,4),sample(1:4,4)),nrow=4)
# finally a rank clock
radial.plot(radmat,rp.type="l",radial.pos=seq(0,20*pi/11.1,length.out=10),
 label.pos=seq(0,20*pi/11.1,length.out=10),start=pi/2,clockwise=TRUE,
 labels=2001:2010,radial.lim=c(0.2,4),main="Rank clock")
legend(-1.7,4,c("Black","Red","Green","Blue"),col=1:4,lty=1)
par(xpd=oldpar$xpd,mar=oldpar$mar,pty=oldpar$pty)
# reset the margins
par(mar=c(5,4,4,2))
```

`radial.plot.labels` *Display labels on a circular grid*

Description

'`radial.plot.labels`' displays a labels on a circular plot produced by one of the radial.plot family of functions.

Usage

```
radial.plot.labels(lengths,radial.pos=NULL,units="radians",radial.lim=NULL,
 start=0,clockwise=FALSE,labels,adj=NULL,pos=NULL,boxed.labels=FALSE,...)
```

Arguments

lengths	numeric data vector. Magnitudes will be represented as the radial positions of symbols, line ends or polygon vertices.
radial.pos	numeric vector of radial positions. These will be converted to radians if the 'units' argument is not "radians".
units	The units of 'radial.pos' may be degrees or 24 hour clock positions. If 'units' is "polar" or "clock24" respectively, the values of radial.pos will be converted into radians.
radial.lim	Optional radial limits for the circular plot. These must be the same as the radial limits of the original plot.
start	The zero position on the plot in the units of 'radial.pos'.
clockwise	Whether to increase angles clockwise rather than the default counterclockwise.
labels	text labels to display on the plot.
adj	Text justification as in the 'text' function.
pos	Text position as in the 'text' function.
boxed.labels	Whether to use 'boxed.labels' or 'text'.
...	additional arguments passed to 'boxed.labels' or 'text'.

Details

Don't confuse this function with the 'radial.labels' argument in the radial.plot function. This labels the values rather than the grid.

Value

nil

Author(s)

Jim Lemon

See Also

text

Examples

```
testlen<-c(rnorm(10)*2+5)
# do the labels in clock24 units
testpos<-c(6.74,8.3,10.55,12.33,13.75,15.9,17.15,19.36,21.02,23.27)
oldpar<-clock24.plot(testlen,testpos,main="Test radial.plot.labels",
 rp.type="s",point.symbols=3,point.col="green")
radial.plot.labels(testlen,testpos,units="clock24",labels=LETTERS[1:10],
 pos=3,col="red")
testangle<-c(25,42,67,94,128,173,191,234,268,307)
# now a polar plot
polar.plot(testlen,testangle,main="Test radial.plot.labels",rp.type="p",
 poly.col="green")
radial.plot.labels(testlen,testangle,units="polar",labels=LETTERS[1:10])
# reset par
par(oldpar)
```

---

radialtext *Display text in a radial line*

Description

Displays a string in a radial line, rotating it to flow in the radial direction and optionally scaling each letter's size according to its distance from the center.

Usage

```
radialtext(x, center=c(0,0), start=NA, middle=1, end=NA, angle=0,
 deg=NA, expand=0, stretch=1, nice=TRUE, cex=NA, ...)
```

Arguments

x	A character string.
center	The center of the circular area in x/y user units.
start	The starting distance of the string from the center in x/y user units.
middle	The middle distance of the string from the center in x/y user units.
end	The ending distance of the string from the center in x/y user units.
angle	The angular position of the string in radians.
deg	The angular position of the string in degrees (takes precedence if not NA).
expand	Size expansion factor for characters, used only if 'start' specified.
stretch	How much to stretch the string for appearance, 1 for none.
nice	TRUE to auto-flip text to keep it upright, FALSE to let it be upside down.
cex	The overall character expansion factor, NA for par("cex").
...	Additional arguments passed to 'text'.

Details

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 a value for 'middle' or 'end'. Likewise, a value for 'end' will override a value for 'middle'. Also, a value for 'deg' overrides any value passed to 'angle'. If 'expand' is 0, all characters will be the same size, while a value of 1 will scale characters so that one that is twice as far from the center will be twice as large. Negative values are permitted too, but 'expand' is only used if 'start' was specified.

Value

nil

Author(s)

Ted Toal

See Also

text, arctext

Examples

```
plot(0, xlim=c(1,5), ylim=c(1,5), main="Test of radialtext",
 xlab="", ylab="", type="n")
points(3, 3, pch=20)
radialtext("uncooked spaghetti", center=c(3,3),
 col="blue")
radialtext("uncooked spaghetti", center=c(3,3),
 start=1.2, angle=pi/4, cex=0.8)
radialtext("uncooked spaghetti", center=c(3,3),
 middle=1.2, angle=pi/4+0.1, cex=0.8)
radialtext("uncooked spaghetti", center=c(3,3),
 end=1.2, angle=pi/4+0.2, cex=0.8)
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.5, deg=135, cex=0.8, col="green")
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.5, deg=145, cex=0.8, stretch=2)
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.5, deg=20, expand=0, col="red")
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.5, deg=250, expand=0.35)
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.75, deg=225, expand=1, col="gold")
radialtext("uncooked spaghetti", center=c(3,3),
 start=0.5, deg=325, expand=-0.25, cex=2)
```

raw.means.plot *raw.means.plot: Raw-Means Plots for Experimental Designs*

Description

raw.means.plot is a function for visualizing results of experimental designs with up to two factors. It plots both raw data (background) and factor/cell means (foreground) to provide a more accurate visualization of the underlying distribution.

Usage

```
raw.means.plot(data, col.offset = 2, col.x = 3, col.value = 4, na.rm = FALSE,
 avoid.overlap = c("y", "x", "both"), y.factor = 1, y.amount = NULL,
 x.amount = 0.05, pch = 21:25, lty = 1:5, bg.b.col = "darkgrey",
 bg.f.col = NULL, fg.b.col = "black",fg.f.col = "black", type = "o",
 pt.cex = 1, lwd = 1, xlab = "", ylab = "", ylim, max.offset = 0.2,
 xaxis = TRUE, x.labels, xaxt = "n", plot = TRUE, legend = TRUE, mar = NULL,
 reset.mar = TRUE, l.pos, yjust = 0.5, l.bty = "n", l.adj = c(0, 0.5), ...)

raw.means.plot2(data, col.id, col.offset, col.x, col.value,
 fun.aggregate = "mean", ...)
```

Arguments

data	a 'data.frame' in long format (i.e., each datapoint one row, see '\link{reshape}' or the reshape package) that contains at least three columns: one column coding the first factor ('col.offset'), one column coding the second factor ('col.x'), and one column containing the values ('col.value').
col.id	a 'character' scalar, specifiying the name of the column specifying the id column. (only for 'raw.means.plot2')
col.offset	a 'character' or 'numeric' (only 'raw.means.plot') scalar, specifiying either name or number of the column coding the different lines (the offset or first factor).
col.x	a 'character' or 'numeric' (only 'raw.means.plot') scalar, specifiying either name or number of the column coding the x-axis factor. Default is 3.
col.value	a 'character' or 'numeric' (only 'raw.means.plot') scalar, specifiying either name or number of the data column. Default is 4.
na.rm	'logical' indicating whether 'NA' values should be stripped before the computation proceeds. Default is 'FALSE'. Throws an error message if FALSE and NAs are encountered.
avoid.overlap	character. What should happen to datapoints within one cell of the two factors that have the same value.

- '"y"' (the default) jitter is added so that overlapping points are distinguishable on the **y**-axis

- '"x"' jitter is added so that overlapping points are distinguishable on the **x**-axis
- '"both"' jitter is added so that overlapping points are distinguishable on both the **y**- and the **x**-axis.
- anything else. No jitter is added.

`y.factor` 'factor' for controlling the amount of jitter on the y-axis (will be passed to jitter).

`y.amount` 'amount' for controlling the amount of jitter on the y-axis (will be passed to jitter).

`x.amount` 'amount' for controlling the amount of jitter on the x-axis (will be passed to jitter).

`pch` 'pch' values (plot symbols) taken for plotting the data. Note that the same values are taken for raw data and means. see points for more details. Recycled if too short (with warning). Default is 21:25, because those are the only values that can be displayed filled and non-filled. All other values should not be used.

`lty` 'lty' values (line types) for connecting the means. See par for more details. Recycled if too short (with warning). Default is 1:5.

`bg.b.col` background border color: border color of raw data points. Silently recycled. Default: '"darkgrey"'

`bg.f.col` background filling color: fill color of raw data points. Silently recycled. Default: 'NULL'

`fg.b.col` foreground border color: border color of mean data points. Silently recycled. Default: 'black'

`fg.f.col` foreground fill color: fill color for mean data points. Silently recycled. Default: 'black'

`type` same as type in plot. Default: 'o' ("overplotted")

`pt.cex` 'numeric' specifying the 'cex' value used for plotting the points. Default is 1.

`lwd` 'numeric' specifying the 'lwd' value used for plotting the lines. Default is 1.

`xlab` x-axis label. Default: '""'

`ylab` y-axis label. Default: '""'

`ylim` the y-axis limits of the plot. If not specified (the default) will be taken from data so that all raw data points are visible and a warning message is displayed specifying the ylim.

`max.offset` 'numeric'. maximal offset of factor levels from the offset factor ('col.offset') specifying the different lines. The centre of each factor on the x-axis is at full numbers (starting from 1 to ...). The maximum will only be reached if the number of factor levels (from 'col.offset') is even. Default: 0.2.

`xaxis` 'logical' value indicating whether or not the x-axis should be generated by 'raw.means.plot'. If 'TRUE', labels for the x-axis will be taken either from the unique values of 'col.x' or can be specified with 'x.labels'.

`x.labels` 'character' vector specifiying 'col.x' levels. Only relevant if 'xaxis=TRUE'. Then, the values given here will be displayed at the x-axis for each factor level of 'col.x'.

xaxt	A character which specifies whether ot not the x-axis should be plotted by the call to plot function. Interfers with the aforementioned 'xaxis' argument and the automatic 'xaxis' function by 'raw.means.plot'. Just there for completeness. Default '"n"' (and should not be changed).
plot	'logical'. Should the 'raw.means.plot' be drawn or not. If 'TRUE' (the default) plot will be drawn. If 'FALSE' only the legend will be drawn (if 'legend = TRUE') See details.
legend	'logical' indicating whether or not 'raw.means.plot' should automatically add a legend on the right outside the plot area indicating which line and points refer to which 'col.offset' factor levels. Default is 'TRUE'.
mar	'NULL' or 'numerical' vector of length 4 indicating the margins of the plot (see par). If 'NULL' (the default) the right margin (i.e., 'par("mar")[4]') will be (imperfectly) guessed from the 'col.offset' factors for placing the legend right to the plot. If length is four this value will be taken. Ignored for 'plot = FALSE'.
reset.mar	'logical' indicating if the margins ('mar') shall be resetted after setting internally. Will be ignored if 'legend = FALSE'. Default is 'TRUE' and should not be changed (especially with 'plot = FLASE').
l.pos	'numeric' vector of length 2 indicating the position of the legend. If not specified automatically determined. See details.
yjust	how the legend is to be justified relative to the legend y location. A value of 0 means top, 0.5 means centered and 1 means bottom justified. Default is 0.5.
l.bty	the type of box to be drawn around the legend. The allowed values are '"o"' and '"n"' (the default).
l.adj	'numeric' of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when labels are plotmath expression. see legend and plotmath for more info.
...	further arguments which are either passed to plot or legend (or 'raw.means.plot' for 'raw.means.plot2'). The following arguments are passed to legend, all others are passed to plot: '"fill","border","angle","density","box.lwd","box.lty","box.col","p
fun.aggregate	Function or function name used for aggregating the data across the two factors. Default is '"mean"'. (only for 'raw.means.plot2')

Details

'raw.means.plot2' is probably the more useful function, as it allows for using a data.frame with more than two-factors and aggregates across the other factors, but needs a column specifying the experimental unit (e.g., participant).

'raw.means.plot' is basically an advanced wrapper for two other functions: plot and (if 'legend=TRUE') legend. Furthermore, raw data is plotted with a call to points and the means with a call to lines.

You can use 'raw.means.plot' to plot only a legend by setting 'plot = FALSE' and 'legend = TRUE'. Then, 'raw.means.plot' will draw an invisible plot with 'xlim = c(0,10)' and 'ylim = c(0,10)' and place the legend on this invisible plot. You can specify 'l.pos' to position the legend, otherwise it will be plotted at 'c(5,5)' (i.e., in the middle of the plot). Note that 'xpd = TRUE' in the call to 'legend' (see par).

Value

Nothing. This function is invoked for its side effects.

Author(s)

Henrik Singmann (<henrik.singmann@psychologie.uni-freiburg.de>) with ideas from Jim Lemon

See Also

add.ps can be used in addition to'raw.means.plot' to compare the factors at each x-axis position, by adding p-values from t-tests to the x-axis.

Examples

```
x <- data.frame(id = 1:150, offset = rep(c("Group A", "Group B", "Group C"),
 each = 50), xaxis = sample(c("A", "B", "C", "D"),150, replace = TRUE),
 data = c(rnorm(50, 10, 5), rnorm(50, 15,6), rnorm(50, 20, 5)))

raw.means.plot(x)

raw.means.plot(x, main = "Example", ylab = "Values", xlab = "Factor",
 title = "Groups")

raw.means.plot(x, "offset", "xaxis", "data")

raw.means.plot(x, "xaxis", "offset", "data")

raw.means.plot(x, 3, 2, 4)

# different colors:
raw.means.plot(x, main = "Example", ylab = "Values", xlab = "Factor",
 title = "Groups", fg.f.col = c("red","blue", "green"))

x2 <- data.frame(id = 1:150, offset = rep(c("Group A", "Group B", "Group C"),
 each = 50), xaxis = sample(c("A", "B", "C", "D"),150, replace = TRUE),
 data = c(rnorm(50, 10, 5), rnorm(50, 15,6), rnorm(50, 20, 5)))

layout(matrix(c(1,2,3,3), 2,2,byrow = TRUE), heights = c(7,1))
raw.means.plot(x, main = "Data x1", ylab = "Values", xlab = "Factor",
 legend = FALSE, mar = c(4,4,4,1)+0.1)
raw.means.plot(x2, main = "Data x2", ylab = "Values", xlab = "Factor",
 legend = FALSE, mar = c(4,4,4,1)+0.1)
raw.means.plot(x2, plot = FALSE, title = "Groups")


y <- data.frame(id = 1:300, offset = rep(1, 300),
 axis = sample(LETTERS[1:6],300, replace = TRUE), data = c(rnorm(100, 1),
 rnorm(100), rnorm(100,1)))

par(mfrow = c(2,2))
```

```
raw.means.plot(y, legend = FALSE)

raw.means.plot(y, type = "p", legend = FALSE)

raw.means.plot(y, type = "l", legend = FALSE)

raw.means.plot(y, 3, 2, x.labels = "one group only")


# Example with overlapping points
z <- data.frame (id = 1:200, offset = rep(c("C 1", "C 2"), 200),
 axis = sample(LETTERS[1:4], 200, replace = TRUE),
 data = sample(1:20, 200, replace = TRUE))

# x versus y jitter
par(mfrow = c(2,2))
raw.means.plot(z, avoid.overlap = "none", main = "no-jitter")
raw.means.plot(z, main = "y-axis jitter (default)")
raw.means.plot(z, avoid.overlap = "x", main = "x-axis jitter")
raw.means.plot(z, avoid.overlap = "both", main = "both-axis jitter")


# y-axis jitter (default)
par(mfrow = c(2,2))
raw.means.plot(z, avoid.overlap = "none", main = "no jitter")
raw.means.plot(z, y.factor = 0.5, main = "smaller y-jitter")
raw.means.plot(z, main = "standard y-jitter")
raw.means.plot(z, y.factor = 2, main = "bigger y-jitter")


# x-axis jitter (default)
par(mfrow = c(2,2))
raw.means.plot(z, avoid.overlap = "none", main = "no jitter")
raw.means.plot(z, avoid.overlap = "x", x.amount = 0.025,
 main = "smaller x -jitter")
raw.means.plot(z, avoid.overlap = "x", main = "standard x-jitter")
raw.means.plot(z, avoid.overlap = "x", x.amount= 0.1,
 main = "bigger x-jitter")



## Not run:

#The examples uses the OBrienKaiser dataset from car and needs reshape.
require(reshape)
require(car)
data(OBrienKaiser)
OBKnew <- cbind(factor(1:nrow(OBrienKaiser)), OBrienKaiser)
colnames(OBKnew)[1] <- "id"
OBK.long <- melt(OBKnew)
OBK.long[, c("measurement", "time")] <-
 t(vapply(strsplit(as.character(OBK.long$variable), "\\."),  "[", c("", "")))
```

```
raw.means.plot2(OBK.long, "id", "measurement", "gender", "value")

raw.means.plot2(OBK.long, "id", "treatment", "gender", "value")

# also use add.ps:
# For this example the position at each x-axis are within-subject comparisons!
raw.means.plot2(OBK.long, "id", "measurement", "gender", "value")
add.ps(OBK.long, "id", "measurement", "gender", "value", paired = TRUE)
 #reference is "fup"

raw.means.plot2(OBK.long, "id", "measurement", "gender", "value")
add.ps(OBK.long, "id", "measurement", "gender", "value", ref.offset = 2,
 paired = TRUE) #reference is "post"

# Use R's standard (i.e., Welch test)
raw.means.plot2(OBK.long, "id", "treatment", "gender", "value")
add.ps(OBK.long, "id", "treatment", "gender", "value",
 prefixes = c("p(control vs. A)", "p(control vs. B)"))

# Use standard t-test
raw.means.plot2(OBK.long, "id", "treatment", "gender", "value")
add.ps(OBK.long, "id", "treatment", "gender", "value", var.equal = TRUE,
 prefixes = c("p(control vs. A)", "p(control vs. B)"))


## End(Not run)
```

`rectFill` *Draw a rectangle filled with symbols*

Description

Draws a rectangle on the current figure filled with arbitrary symbols.

Usage

```
rectFill(x1,y1,x2,y2,fg=par("fg"),bg=par("bg"),xinc=NA,yinc=NA,
 pch=1,pch.cex=1,pch.col=par("fg"),...)
```

Arguments

`x1,y1,x2,y2`	Rectangle limits as in '`rect`'.
`fg`	Foreground color
`bg`	Background color
`xinc,yinc`	The x and y increments of spacing for the symbols.
`pch`	Which symbol to use

pch.cex	Character expansion for the symbols.
pch.col	Color(s) for the symbols.
...	Additional arguments to 'points' for the symbols.

Details

'rectFill' draws a rectangle and fills the rectangle with the symbols requested. It is probably most useful as a substitute for fill colors in a black and white environment.

Value

nil

Author(s)

Jim Lemon

See Also

rect, points

Examples

```
plot(1:7,type="n",xlab="",ylab="",main="Test of rectFill")
rectFill(1:6,1:6,2:7,2:7,bg=2:7,pch=c("+","*","o",".","#","^"),
 xinc=c(0.2,0.1,0.2,0.1,0.2,0.2),yinc=c(0.2,0.1,0.2,0.1,0.2,0.2),
 pch.col=1:6)
barp(matrix(runif(9),nrow=3),main="Black and white bar plot",pch=1:3)
```

---

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)
```

---

ruginv *Add an Inverse Rug to a Plot*

Description

Adds a *rug* representation (1D plot) of the data to the plot, but with the coloring inverted.

Usage

```
ruginv(x,ticksize=0.03,side=1,lwd=0.5,col=par("fg"),col.ticks="white",
quiet=getOption("warn") < 0,...)
```

Arguments

x	A numeric vector.
ticksize	The length of the ticks making up the 'rug'. Positive lengths produce inward ticks.
side	On which side of the plot box the rug will appear. Usually 1 (bottom) or 3 (top).
lwd	The line width of the ticks.
col	Color of the background of the ticks.
col.ticks	The color of the ticks.
quiet	Logical indicating if there should be a warning about clipped values.
...	Further arguments passed to polygon when plotting the background for the ticks.

Author(s)

Peter Solymos

See Also

rug

Examples

```
require(stats)
plot(density(faithful$eruptions,bw=0.15))
ruginv(faithful$eruptions,ticksize=-0.05)
ruginv(jitter(faithful$eruptions,amount=0.01),side=3,col="lightblue")
```

---

seats *Arrange N seats in M semicircular rows*

Description

Compute seat positions in a semicircular parliament

Usage

```
seats(N, M, r0 = 2.5)
```

Arguments

N	Total number of seats.
M	Number of semicircular arcs on which to distribute the seats.
r0	Radius of the inner arc in user units.

Value

A data frame including:

x	The x positions of the seats to be plotted on semi-circular arcs.
y	The y positions of the seats to be plotted on semi-circular arcs.
r	The row numbers for each seat.
theta	The angle of each seat, going from pi to zero radians.

Author(s)

Duncan Murdoch and Barry Rowlingson

See Also

election

`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 $scale * n^pow$, 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,
 stacklabels=TRUE,firstcall=TRUE,col=NULL,border=NA,toplab=NULL,base.cex=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.
`stacklabels`	Whether to display the names of the dataframe beneath the stacked rectangles.
`firstcall`	A flag for the function - do not alter this.
`col`	Optional fill colors for the rectangles. See Details
`border`	Color for border around the rectangles. See details
`toplab`	Optional labels to display a the top of each stack.
`base.cex`	The base character expansion for the labels.
`...`	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, which will propagate across subsequent stacks.

The user can simply pass the data frame, which should only contain columns that are hierarchical categories (example 1). The colors will probably not be ideal. The user can pass the same colors for the all levels (example 2). If this is done, '`sizetree`' will try to match colors to categories when the number of categories is diminishing (e.g. some levels are missing in the sub-categories) and the columns of '`x`' are factors with the same levels in the same order. This will work if the category labels are the same in each level, but remember to add the names to the colors before passing them to the function. This will not work if there are more categories in the lower levels. If '`col`' is a list, this is not done, and the user will have to work out the correct colors for each level. This is particularly important when the category labels and the number of categories is different in different levels (example 3).

In some sizetrees, the subcategory counts are very low compared to the overall number of data objects. This results in rectangles that are very thin vertically. One way to get better legibility of the labels is to use dark colors for the rectangles, so that the labels are white, and no borders (set '`border`' to NA). The user can also select only part of the data frame '`x`' to expand sections of the sizetree as in the last example.

The labels are sized to fit the vertical extent of the bars. However, it is possible that the labels may extend horizontally beyond the bar(s). The '`base.cex`' argument can be used to shrink the labels if this happens. Remember that '`base.cex`' will shrink all the labels, not just the ones that are too wide.

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

Examples

```
cat1<-factor(sample(c("None","Low","Medium","High","Extreme"),40,TRUE),
 levels=c("None","Low","Medium","High","Extreme"))
cat2<-factor(sample(c("None","Low","Medium","High"),40,TRUE),
 levels=c("None","Low","Medium","High"))
cat3<-factor(sample(c("None","Low","High"),40,TRUE),
 levels=c("None","Low","High"))
hcats<-data.frame(cat1,cat2,cat3)
# throw in a few NAs
hcats$cat1[10]<-NA
hcats$cat2[c(15,20)]<-NA
hcats$cat3[c(11,14,25)]<-NA
# first let sizetree work out the colors
sizetree(hcats,main="Sizetree with automatic colors")
# now see what happens with a list of the same colors for each level
```

```
bhcol<-c("#ff8080","#dddd80","#80ff80","#0000ff","#80dddd")
sizetree(hcats,col=list(bhcol,bhcol,bhcol),
 main="Sizetree with the same colors each level")
# finally, specify different colors for categories with different labels
sexhaireye<-data.frame(sex=factor(sample(c("Male","Female"),50,TRUE)),
 hair=factor(sample(c("Blond","Red","Brown","Black"),50,TRUE)),
 eye=factor(sample(c("Gold","Green","Blue"),50,TRUE)))
shecol<-list(c("pink","lightblue"),c("#000000","#dddd00","#886600","#ee8800"),
 c("blue","gold","green"))
sizetree(sexhaireye,main="Sex, hair and eye color",
 col=shecol,toplab=c("Sex","Hair color","Eye color"))
# now expand the female part of the sizetree
sizetree(sexhaireye[sexhaireye[,1]=="Female",],
 main="Sex, hair and eye color (Females only)",
 col=shecol,toplab=c("Sex","Hair color","Eye color"))
```

`size_n_color` *Display circles with specified size and color*

Description

Display a plot of circles at specified locations, each with a specified size and color.

Usage

```
size_n_color(x=NULL,y,size,sizefun="sqrt",col,main="",
 xlim=NA,xlab="",xat=NULL,xaxlab=NULL,xcex=1,xlas=0,xgrid=FALSE,
 ylim=NA,ylab="",yat=NULL,yaxlab=NULL,ycex=1,ylas=1,ygrid=TRUE,
 mar=c(5,4,4,2),boxit=TRUE,add=FALSE,...)
```

Arguments

`x,y`	Vectors or matrices of x and y positions for the symbols.
`size`	Sizes for the symbols expressed as numbers.
`sizefun`	The function to use for transforming the values to radii of circles. Square root gives areas proportional to the values.
`col`	Colors for the symbols (see Details).
`main`	Title for the plot.
`xlim,ylim`	Explicit x and y limits for the plot.
`xlab,ylab`	Labels for the x and y axes.
`xat,yat`	Where to place the ticks and tick labels on the axes.
`xaxlab,yaxlab`	Tick labels for the x and y axes.
`xcex,ycex`	Character expansions for the axis tick labels.
`xlas,ylas`	Orientation for the axis tick labels (see 'par').
`xgrid,ygrid`	Whether to display a grid along the x or y direction.

mar	Margins for the plot (see Details).
boxit	Whether to draw a box around the plot.
add	Whether to draw a new plot (FALSE) or add symbols to an existing plot (TRUE).
...	Additional arguments passed to 'plot'.

Details

'size_n_color' plots circles centered on the 'x' and 'y' coordinates. The size and color of the circles may also be specified individually, allowing four dimensions of variation to be displayed on the plot.

'size_n_color' may also be used to display a "visual table" as in the second example. Here the x and y coordinates are used to associate the symbols with two categorical variables, underlying cause of death and year of observation. If the x values are not passed to the function, it will try to space out the circles evenly in a representation of the matrix. If the matrix is not square, use a plotting device that has about the same proportion of height and width as the matrix.

Value

nil

Author(s)

Jim Lemon

See Also

plot, points, par

Examples

```
meantemp<-c(19,22,25,29,21,20,16,27,23,26)
totalrain<-c(174,152,196,120,177,183,92,153,161,85)
numpumpkin<-c(53,47,61,63,38,42,48,71,66,29)
meanwt<-c(1.5,2.3,2.8,1.9,2.4,1.8,2.6,2.2,1.7)
size_n_color(meantemp,totalrain,meanwt/5,NA,xlim=c(15,30),
 color.scale(numpumpkin,c(0.8,0),c(0.8,1),0),
 xlab="Temperature (degrees C)",ylab="Rainfall (mm)",
 main="Number and weight of pumpkins by temperature and rainfall",
 xat=seq(15,30,by=5),yat=seq(80,200,by=20))
color.legend(15,55,18.5,60,seq(40,70,by=10),
 rect.col=color.scale(seq(40,70,by=10),c(0.8,0),c(0.8,1),0))
points(15:18,rep(126,4),cex=seq(1.5,3.0,by=0.5))
text(15:19,rep(134,5),c("1.5","2.0","2.5","3.0","kg"))
par(xpd=TRUE)
text(13.5,60,"Number of\npumpkins")
par(xpd=FALSE)
# now display a "visual table" of delayed registrations by underlying cause of
# death and year of observation. The sizes of the circles represent the log of
# the number of deaths and the colors represent the percentage of deaths that
# occurred in the year prior to registration or earlier
```

```
data(death_reg)
size_n_color(x=matrix(rep(1996:2010,each=22),nrow=22),
 y=matrix(rep(1:22,15),nrow=22),size=t(death_reg[[1]])/200,
 col=color.scale(t(death_reg[[2]]),c(0,0.8,1),c(1,0.2,0),0),
 ylim=c(1,22),main="Delayed registrations by ICD chapter",
 xlab="Year",xaxlab=1996:2010,xat=1996:2010,xcex=0.8,
 yaxlab=colnames(death_reg[[1]]),ycex=0.8,ygrid=TRUE,mar=c(5,6,4,2))
color.legend(1994,-3.5,2000,-2.5,seq(0,50,by=10),cex=0.8,
 rect.col=color.scale(seq(0,50,by=10),c(0,0.8,1),c(1,0.2,0),0))
par(xpd=TRUE)
text(1993.4,-2.5,"Pct.\nslow",cex=0.8)
par(xpd=FALSE)
```

`sliceArray` *Slice an array*

Description

Slices one dimension from an array by taking one element from the first dimension.

Usage

```
sliceArray(x,slice)
```

Arguments

`x`	An array
`slice`	The index of the slice to take from the first dimension of the array.

Details

'`sliceArray`' builds an extractor string containing the value of '`slice`' as the first element and as many commas as needed to match the dimensions of the array. It then applies the extractor function to '`x`' and returns the result. Note how the array "slice" swaps dimensions in the example.

Value

The desired slice of the array.

Author(s)

Jim Lemon

See Also

array

Examples

```
a1<-array(1:27,dim=c(3,3,3))
a1
sliceArray(a1,2)
```

`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`'.

Note

For more R functions that transform numeric values into colors or produce colors that can be used to represent values, see the **colourschemes** package.

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.// https://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/survey/class/taxonomy/?cid=nrcs142p2_053577

U.S. Department of Agriculture, Natural Resources Conservation Service, 2007. *Soil Texture Calculator*// https://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/survey/class/taxonomy/?cid=nrcs142p2_053577

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', 'spread.lab (TeachingDemos)'

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=NULL,staxx=FALSE,stack=FALSE,
 axis2=TRUE,axis4=TRUE,padj=0,...)
```

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. If NULL, 'rainbow' will be called to generate the colors. If NA, the polygons will not be filled.
staxx	Whether to call 'staxlab' to stagger the x axis labels.
stack	Whether to stack the successive values on top of each other.
axis2	Whether to display the left ordinate on the plot.
axis4	Whether to display the right ordinate on the plot.
padj	Vertical justfication of the x axis labels, defaulting to "top". Can be a vector with an element for each label.
...	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".

The options for 'axis2' and 'axis4' can be used to produce panel plots. See the last example.

Value

nil

Author(s)

Jim Lemon and Thomas Petzoldt (waterfall plot option) - thanks to Phil Novack-Gottshall for the mismatched x and y fix

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)
layout(matrix(1:2,nrow=1))
oldmar<-par(mar=c(5,4,4,0))
stackpoly(rev(sort(testx-mean(testx))),
 main="Waterfall Plot (x-mean)",xat=seq(10,90,by=10),
 xlab="Index",ylab="Value",lwd=3,col="green",border="black",
 axis4=FALSE)
ylim<-par("usr")[3:4]
par(mar=c(5,0,4,4))
stackpoly(rev(sort((testx-mean(testx))/sd(as.vector(testx)))),
 ylim=ylim,main="Waterfall Plot ((x-mean)/sd)",xat=seq(10,90,by=10),
 xlab="Index",lwd=3,col="lightblue",border="black",axis2=FALSE)
par(oldmar)
```

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,cex=par("cex"),prefix="",suffix="",...)
```

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.
cex	The usual character expansion value.
prefix	A prefix to the numbers displayed next to the bars (e.g. $).
suffix	A suffix as for prefix (e.g. %).
...	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 '`prefix`' and '`suffix`' arguments allow the user to specify units for the numbers displayed next to the bars. If a single value is passed, all numbers will get the same prefix or suffix. Different prefixes or suffixes for each number can be passed as vectors.

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 (staircase.plot)",
 total.col="gray",inc.col=2:4,bg.col="#eeeebb",direction="s")
```

staircasePlot *Display a staircase plot*

Description

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

Usage

```
staircasePlot(heights,totals=NA,labels=NULL,halfwidth=0.3,main="",mar=NA,
stair.info=list(total.col="blue",inc.col=NA,border=par("fg")),bg.col=NA,
direction="e",las=1,display.height=TRUE,stagger=FALSE,cex=par("cex"),
prefix="",suffix="",...)
```

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.
stair.info	A list of arguments for the bars including color(s) for the bars representing successive totals, and increments and the border color.
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.
cex	The usual character expansion value.
prefix	A prefix to the numbers displayed next to the bars (e.g. $).
suffix	A suffix as for prefix (e.g. %).
...	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 '`prefix`' and '`suffix`' arguments allow the user to specify units for the numbers displayed next to the bars. If a single value is passed, all numbers will get the same prefix or suffix. Different prefixes or suffixes for each number can be passed as vectors.

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")
staircasePlot(sample_size,totals,labels,
 main="Acquisition of the sample (staircasePlot)",
 total.col="gray",inc.col=2:4,bg.col="#eeeebb",direction="s")
```

starPie *A pie-like graphic object*

Description

Display a polygon with each sector proportional to a vector of numeric values.

Usage

```
starPie(x,y,radext,values,maxval=NA,border=par("fg"),col=NA,prop.area=FALSE,
 label="",labelpos=1)
```

Arguments

x,y	The coordinate position for the center of the starPie.
radext	The maximum distance from the center of the starPie to one vertex of the polygon.
values	A vector of numeric values.
maxval	A maximum value for scaling the values to the radius. If NA, the maximum value in 'values' will be used.
border	The color to use for the borders of the polygon sectors.
col	The color(s) to use for the fills of the polygon.
prop.area	Whether to scale the values to the area (TRUE) or the radial extent (FALSE) of the polygon sectors.
label	Optional text labels for the starPies.
labelpos	Positions of the labels relative to the starPies.

Details

'starPie' displays a polygon centered on the 'x,y' position having sectors of equal angular extent. The radial extent of each sector is proportional to the values in the numeric vector 'lengths'. If the 'prop.area' argument is TRUE, the proportion is based on the area of the sector, and if 'prop.area' is FALSE, the proportion is on the radial extent. As the function is intended to exaggerate the differences between different starPies, the default produces sectors proportional to the squares of the 'lengths'.

'starPie' is intended to display a visual analog of the relative value of matched attributes of a number of similar objects or groups. Thus objects having similar attributes will produce similar looking starPies. When constructing such a matrix, it is necessary for 'maxval' to be specified, usually as the overall maximum value in any of the attribute value vectors. If 'maxval' is not specified in such a situation, only the relative values within each vector will determine the radial extents of each starPie. There appears to be no reason to have different sector colors for different objects, but the user can display more than one set of starPies on a plot with different sector colors if necessary.

'starPie' calls 'getYmult' to automatically adjust for both the aspect and coordinate ratio of the plot.

Value

nil

Author(s)

Jim Lemon

Examples

```
date_mat<-data.frame(sex=rep(c("M","F"),each=10),
 names=c("Abe","Bob","Col","Dave","Eddie","Frank","Geoff","Harry","Igor","Jack",
 "Alice","Betty","Clare","Dora","Eva","Fran","Grace","Hilda","Iris","Joan"),
 eating=sample(0:100,20),dancing=sample(0:100,20),movies=sample(0:100,20),
 reading=sample(0:100,20),travel=sample(0:100,20))
plot(0,xlim=c(0.5,10.5),ylim=c(0,3),type="n",axes=FALSE,xlab="",ylab="Sex",
 main="Date matching matrix")
par(xpd=TRUE)
legend(0.7,-0.3,c("Eat out","Dance","Movies","Read","Travel"),fill=rainbow(5),
 ncol=5)
par(xpd=FALSE)
box()
axis(2,at=c(0.9,2.4),labels=c("Male","Female"))
starPie(x=rep(1:10,2),y=rep(c(0.9,2.4),each=10),radext=0.5,
 values=as.matrix(date_mat[,3:7]),label=as.character(date_mat[["names"]]))
```

---

staxlab *Place staggered or angled labels on an axis*

Description

Places labels on an axis in a regular staggered order or at an angle

Usage

```
staxlab(side=1,at,labels,nlines=2,top.line=0.5,line.spacing=0.8,
 srt=NULL,ticklen=0.03,adj=1,...)
```

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.
srt	Text rotation.

ticklen	Proportion of plot height in user units to place text below the plot.
adj	horizontal adjustment of the labels.
...	Additional arguments to be passed to 'mtext' or 'text'.

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.

If 'srt' is not NULL, the labels will be rotated 'srt' degrees and placed below the plot. This method will only place labels at the bottom. Note that this option only works on the lower and left axes.

Author(s)

Jim Lemon (thanks to Tim Elwell-Sutton for the log axis fix)

See Also

mtext

Examples

```
x<-rnorm(12)
plot(x,axes=FALSE)
box()
months<-c("January","February","March","April","May","June",
 "July","August","September","October","November","December")
staxlab(1,1:12,months)
plot(x,axes=FALSE)
box()
staxlab(1,1:12,months,srt=45)
ylabels<-round(seq(min(x),max(x),length.out=10),3)
staxlab(2,ylabels,ylabels,srt=45)
```

---

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

---

sumbrk *Count specified values in a vector*

Description

Counts the number of values in a vector that are equal to a specified value.

Usage

```
sumbrk(x,trueval=TRUE,na.rm=TRUE)
```

Arguments

x	a character, factor or numeric vector.
trueval	the value to be matched in 'x'.
na.rm	whether to remove NA values.

Details

'sumbrk' counts the values in 'x' matching a specified value. It is mainly to allow these sums to be calculated in the 'brkdnNest' function.

Value

nil

Author(s)

Jim Lemon

See Also

brkdnNest

Examples

```
sumbrk(sample(LETTERS,100,TRUE),trueval="M")
```

`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="Standard deviation",ylab="",main="Taylor Diagram",
 show.gamma=TRUE,ngamma=3,gamma.col=8,sd.arcs=0,
 ref.sd=FALSE,sd.method="sample",grad.corr.lines=c(0.2,0.4,0.6,0.8,0.9),
 pcex=1,cex.axis=1,normalize=FALSE,mar=c(4,3,4,3),...)
```

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).
gamma.col	color to use for the gamma arcs (only with pos.cor=TRUE).
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.
sd.method	Whether to use the sample or estimated population SD.
grad.corr.lines	the values for the radial lines for correlation values (see Details).
pcex	character expansion for the plotted points.
cex.axis	character expansion for the axis text.
normalize	whether to normalize the models so that the reference has a standard deviation of 1.
mar	margins - only applies to the 'pos.cor=TRUE' plot.
...	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=c('l','c','r'), cex=1, leading=0.5, box=TRUE,
adj=c(0,0), font=NULL, vfont=NULL, col=NULL, border=NULL, fill=NA, density=NULL,
angle=45, lty=par("lty"), lwd=par("lwd"), margin=0)
```

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`	x alignment: 'l'=left, 'c'=center, 'r'=right.
`cex`	character expansion
`leading`	inter-line spacing
`box`	whether to draw a box around the text
`adj`	adjustment for x and y position, default is no adjustment, see Details
`font`	text font, see Details
`vfont`	text font, see Details
`col`	text color
`border`	box border color
`fill`	box fill color
`density`	box shading line density, see Details
`angle`	box shading line angle, see Details
`lty`	box border and shading line types, see Details
`lwd`	box border and shading line width, see Details
`margin`	amount to adjust box border in or out. See Details

Details

Draws text in the box by pasting the textlist vector together, splitting it into words, and then adding words to the current line until the line is wide enough before moving on to the next line.

'`margin`' may be a vector of 1, 2, or 4 values, corresponding to adjustment of all borders (1 value), top/bottom and left/right borders (2 values), or bottom/left/top/right borders (4 values). A positive value moves text inwards from specified (x,y) position with border remaining at (x,y), and a negative value moves the border outwards from (x,y) with the text remaining at (x,y).

The '`density`' and '`angle`' arguments have the same behavior as in the '`rect`' function. The '`adj,font`' and vfont arguments have the same behavior as in the '`text`' function. The '`lty`' and '`lwd`' arguments have the same behavior as in the '`lines`' function.

Value

y-position of bottom line of box, or y-position of next line if there is no box.

Author(s)

Ben Bolker. Improvements by Ted Toal.

Examples

```
plot.new()
textbox(c(0,0.2), 1, c("many words","more words","why not?",
 "keep going",rep("and going",10)))
textbox(c(0.3,0.5), 1, c("keep going",rep("and going",10)), cex=0.45,
 col="blue", border="red", fill="#00FFEE80", density=25, angle=60)
textbox(c(0.6,0.8), 1, c("keep going",rep("and going",10)), justify='c', cex=0.6,
 leading=1, font=4, border="gold", lty=2, lwd=4, margin=0.025)
textbox(c(0.6,0.8), 0.5, c("keep going",rep("and going",10)), justify='r', cex=0.7,
 col="purple", font=3, border="green", margin=-0.025)
lines(c(0,1), c(1,1), col="red", lty=2)
lines(c(0,1), c(0.5,0.5), col="red", lty=2)
```

---

thigmophobe *Find the direction away from the closest point*

Description

Find the direction away from the closest point

Usage

```
thigmophobe(x,y=NULL,names=seq_along(z),xlog=par("xlog"),ylog=par("ylog"),
 usr=par("usr"),pin=par("pin"),eps=.Machine$double.eps,pi=base::pi)
```

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.
names	Names for the vector of directions.
xlog,ylog	Flags for logarithmic axes. See Note.
usr	The extent of the plot in user units.
pin	Extent of the plot in inches.
eps	smallest number that can be represented on the system.
pi	value of pi.

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

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

The 'plot.span', 'xlog' and 'ylog' arguments were added to allow 'thigmophobe' to be used outside of base graphics.

Author(s)

Bill Venables

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'.

'thigmophobe' will fail with only two labels, as it can't figure out the nearest neighbors. If you really want to use this with two labels, just eyeball the plot and work out in which direction the labels will go. Then pass the directions to 'thigmophobe.labels' as the 'text.pos' argument. When all else fails, look to sampplaceLabels.

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 (thanks to Stephen Milborrow for finding the single point bug and Erik Aronesty for finding the two point problem.)

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
thigmophobe.labels(x,y,plotlabels,col=c(2:6,8:12),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.fill *Triangle plot fill*

Description

Fill a triangle plot with smaller triangles.

Usage

```
triax.fill(col)
```

Arguments

col	List of colors (see Details).

Details

In order for 'triax.fill' to fill an existing plot that has been created by a call to 'triax.plot', the user must supply a list of fill colors. The first element of the list must begin with at least one value that can be interpreted as a color. The second element must begin with at least three such values, and so on, adding two values for each element of the list. Each list element will be displayed as a row of colored triangles starting at the top of the plot. The number of elements in the list determines the number of rows that will be displayed.

Value

nil

Author(s)

Jim Lemon

See Also

triax.plot,color.scale

Examples

```
# the data will be something like response at different proportions
fillval<-list(0,c(0,0.1,0),c(0,0.1,0.2,0.1,0),
 c(0,0.1,0.2,0.3,0.2,0.1,0),c(0,0.1,0.2,0.3,0.4,0.3,0.2,0.1,0),
 c(0,0.1,0.2,0.3,0.4,0.5,0.4,0.3,0.2,0.1,0),
 c(0,0,0.1,0.2,0.3,0.4,0.5,0.4,0.3,0.2,0.1,0,0),
 c(0,0,0,0.1,0.1,0.2,0.3,0.4,0.3,0.2,0.1,0.1,0,0,0))
# use some method of converting values to colors
fillcol<-sapply(fillval,function(x) {x*10+1} )
oldpar<-triax.plot(main="Test of triax.fill function")
triax.fill(fillcol)
par(oldpar)
```

---

triax.frame *Triangle plot frame*

Description

Display a three axis frame with optional grid.

Usage

```
triax.frame(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

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,triax.fill

Examples

```
triax.plot(main="DEFAULT")
triax.plot(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"),mar=c(5,2,4,2),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.
mar	Margins for the triangle plot.
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)
```

---

tsxpos *Calculate equispaced x positions.*

Description

Calculate equispaced x positions of values that have been plotted with the plot command.

Usage

```
tsxpos(x,xlim,nint)
```

Arguments

x	A vector of numeric values or a time series object created with the ts function.
xlim	Explicit x limits for the x positions.
nint	The number of *intervals* between x positions.

Details

'tsxpos' calculates equispaced x positions for a vector of values or a time series created with the 'ts' command from the **stats** package. It assumes that the default x limits have been used in the existing plot. It adds the appropriate padding if 'par("xaxs")' is "r". It is mainly useful when x axis labels or some other markers are to be added to a time series plot.

A plot device must be open. If the user wishes to specify explicit x limits or the number of intervals (not values), these will override the calculations from the x values.

Value

The calculated x positions in user units.

Author(s)

Jim Lemon (thanks to Prof J.C. Nash for the idea)

Examples

```
# create a vector of numbers
y<-rnorm(28)
par(mfrow=c(2,1),mar=c(6,4,4,2))
plot(y,main="Plot of the values")
# convert it into a time series object
yt<-ts(y,start=2011,frequency=12)
# don't use the default axis
plot(yt,main="Plot of the time series",xaxt="n",xlab="Month")
labelpos<-tsxpos(yt)
# display an axis showing the months only
staxlab(1,labelpos,rep(month.abb,length.out=28))
par(mfrow=c(1,1),mar=c(5,4,4,2))
```

---

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,main="",xlim=NULL,lylim=NULL,rylim=NULL,
mar=c(5,4,4,4),lcol=1,rcol=2,xlab="",lytickpos=NA,ylab="",ylab.at=NA,
rytickpos=NA,rylab="",rylab.at=NA,lpch=1,rpch=2,
type="b",xtickpos=NULL,xticklab=NULL,halfwidth=0.4,axislab.cex=1,
do.first=NULL,xaxt="s",...)
```

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
main	Title for the plot
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	X axis label as in 'plot'
lytickpos	Optional positions for the left axis tick labels.
ylab	Left Y axis label as in 'plot'
ylab.at	Optional position for the left Y axis label
rytickpos	Optional positions for the right axis tick labels.
rylab	Right Y axis label
rylab.at	Optional position for the right Y axis label
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.
do.first	Optional command(s) that will be executed immediately after the blank plot is displayed.
xaxt	Whether to display the x-axis - "n" = no.
...	additional arguments passed to 'plot' and 'points'.

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 overplotted.

Note that more values can be added to the plot using 'points' or 'lines', but remember that these will be plotted relative to the left ordinate.

The 'do.first' argument is useful for adding a background color or grid to the plot as shown in the first two examples.

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. See the second example, which illustrates a method for separating the two series and offsetting the axes.

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' or separate plots 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, Jacob Kasper for the axis character expansion and Ye Lin for finally motivating me to add the do.first argument.)

See Also

plot

Examples

```
xval1 <- seq.Date(as.Date("2017-01-02"),
 as.Date("2017-01-10"), by="day")
xval2 <- seq.Date(as.Date("2017-01-01"),
 as.Date("2017-01-15"), by="day")
going_up<-seq(3,7,by=0.5)+rnorm(9)
going_down<-rev(60:74)+rnorm(15)
twoord.plot(2:10,going_up,1:15,going_down,xlab="Sequence",
```

```
 ylab="Ascending values",rylab="Descending values",lcol=4,
 main="Plot with two ordinates - points and lines",
 do.first="plot_bg();grid(col=\"white\",lty=1)")
axis.Date(1,xval2)
# now separate the lines
twoord.plot(2:10,going_up,1:15,going_down,xlab="Sequence",
 lylim=range(going_up)+c(-1,10),rylim=range(going_down)+c(-10,2),
 ylab="Ascending values",ylab.at=5,rylab="Descending values",
 rylab.at=65,lcol=4,main="Plot with two ordinates - separated lines",
 lytickpos=3:7,rytickpos=seq(55,75,by=5),
 do.first="plot_bg();grid(col=\"white\",lty=1)")
twoord.plot(2:10,going_up,1:15,going_down,xlab="Sequence",
 lylim=range(going_up)+c(-1,10),rylim=range(going_down)+c(-10,2),
 type=c("bar","l"),ylab="Ascending values",ylab.at=5,
 rylab="Descending values",rylab.at=65,
 main="Bars on left axis, lines on right axis",
 lytickpos=3:7,rytickpos=seq(55,75,by=5),
 lcol=3,rcol=4,do.first="plot_bg()")
twoord.plot(2:10,going_up,1:15,going_down,xlab="Sequence",
 lylim=c(-3,8),rylim=c(50,100),type=c("l","bar"),
 ylab="Ascending values",rylab="Descending values",
 lytickpos=3:7,rytickpos=seq(55,75,by=5),ylab.at=5,rylab.at=65,
 main="Lines on left axis, bars on right axis",
 lcol=3,rcol=4,do.first="plot_bg(\"yellow\")")
# histogram with density curve superimposed
xhist<-hist(rnorm(100),plot=FALSE)
xdens<-dnorm(seq(-3,3,by=0.05))
twoord.plot(xhist$mids,xhist$counts,seq(-3,3,by=0.05),
xdens,type=c("bar","l"),lcol=4,rcol=2,ylab="Counts",
rylab="Density",main="Histogram and density curve",
halfwidth=0.2,lylim=c(0,max(xhist$counts)+1),rylim=c(0,0.45),lwd=2)
```

`twoord.stackplot` *Multiple (stack) plot with two ordinates*

Description

Two set of data are plotted on two different ordinate scales.

Usage

```
twoord.stackplot(lx, rx, ldata, rdata, lcol, rcol, ltype, rtype,
border, rylab, lylab, xlab, ..., incrylim=NULL,
halfwidth=0.4, leftfront=FALSE, mar = c(5, 4, 4, 4))
```

Arguments

`lx,rx`	x-values for left/right data.
`ldata,rdata`	data on the left/right y-axes.

lcol, rcol	colors to be used for left/right data.
ltype, rtype	line types to be used for left/right data, see details.
border	color for the border of barplot
rylab,lylab	labels for the left/right y-axes.
xlab	labels for the x-axis.
...	further arguments to be passed to 'plot'.
incrylim	a number to increase the limits of y-axes.
halfwidth	half the width of the bars in user units. The bars are centered on successive integers if no x values are supplied
leftfront	if 'TRUE', plot the left data on the front layer.
mar	optional margin adjustment, defaults to c(5,4,4,4).

Details

'twoord.stackplot' works in the same way as 'twoord.plot' on which it is heavily inspired. The functions let the user plot multiple curve/point or bar plots on the same graph with two different axes. The line type can be one of the following '"l"' for lines, '"p"' for points, '"b"' for both points and line, '"o"' for overplotted, '"bar"' for barplot.

Value

nil

Author(s)

Christophe Dutang

See Also

twoord.plot

Examples

```
# plot data
#

time <- 0:25

A <- 1+1/2*sin(time/2)
B <- A + rnorm(length(A), sd=1/10)
B <- B + rnorm(length(A), sd=1/10)

sizeA <- floor(450*(1 + 1/4*sin(time/2+2))*(1+.1))
sizeB <- 1000-sizeA

C <- (A*sizeA + B*sizeB)/(sizeA+sizeB)
```

```
#typical usage
#

twoord.stackplot(lx=time, rx=time, ldata=cbind(sizeA, sizeB),
rdata=cbind(A, B, C),  lcol=c("grey80", "white"),
rcol=c("blue", "red","black"), ltype="bar", rtype=c("l","p","o"),
border="grey80", lylab="Size", rylab="A,B,C", xlab="Time",
main="a plot", incrylim=2/100)

#add a legend
#

par(xpd=TRUE) #extend the area of plotting
par(new=TRUE) #to add new graph "layers"
plot(0:1, 0:1, type="n", xlab="",ylab="", axes=FALSE) #redo the x/y limits

#first legend
legend(-0.18, 1.2, leg=c("Size A", "Size B"), fill=c("grey80", "white"))
#second legend
legend(.97, -0.08, leg=c("A", "B", "C"), col=c("blue", "red","black"),
pch=c(NA, 19, 19), lty=c(1,NA,1))

par(xpd=FALSE, new=FALSE) #default setting



#reverse the order of plotting
twoord.stackplot(lx=time, rx=time, ldata=cbind(sizeA, sizeB),
rdata=cbind(A, B, C),  lcol=c("grey80", "white"),
rcol=c("blue", "red","black"), ltype="bar", rtype=c("l","p","o"),
border="grey80", lylab="Size", rylab="A,B,C", xlab="Time",
main="a plot", incrylim=2/100, leftfront=TRUE)
```

`valid.n` *Find the number of valid (not NA) values*

Description

Finds the number of valid (not NA) or total values in an object.

Usage

```
valid.n(x,na.rm=TRUE)
```

Arguments

`x`	An object.
`na.rm`	Whether to count all values (FALSE) or only those not NA.

Details

'valid.n' finds the number of valid values of the object if 'na.rm=TRUE'.

Value

The number of valid values or the length of the object.

Author(s)

Jim Lemon

`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"),col=par("fg"))
```

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
col	Color(s) for the arrows.

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)
 vectorcol<-sample(colors(),100)
 vectorField(dir,mag,xpos,ypos,scale=0.8,vecspec="rad",col=vectorcol)
```

---

violin_plot *Display a "violin" plot*

Description

Displays violin plots (rotated kernel density plots on each side of boxplots).

Usage

```
violin_plot(X=rnorm(50),at,add=FALSE,na.rm=TRUE,bw,violin_width,
violin_end_width=0.005,equal_width=TRUE,show_box=TRUE,box_width=0.01,
box_col="black",show_outliers=TRUE,outlier_pch=NA,range=1.5,xlim,ylim,
axes=TRUE,ann=TRUE,xlab="",ylab="",x_axis_labels,main="Violin Plot",
col="red",median_col="white",show_mean=FALSE,mean_pch=19,
mean_pch_col="yellow",...)
```

Arguments

X	A vector or matrix or data frame of numeric values.
at	Horizontal position(s) for the violin plot(s).
add	Whether this violin should be added to an existing plot.
na.rm	Remove NA values. Passed to functions such as 'boxplot' or 'density'.
bw	Vector or bandwidth values for 'density'. Will be recycled. If not provided then will be calculated using 'bw.nrd0'.
violin_width	Multiplier to scale the width of the 'violin'.
violin_end_width	Multiplier to scale the width of the ends of the violin.
equal_width	Should all violin widths be equal?
show_box	Whether to display the box.
box_width	Multiplier for the width of internal boxes.
box_col	Fill color for the internal rectangle.
show_outliers	Whther to display outliers as points.
outlier_pch	Symbol for displaying outliers.
range	Passed to 'boxplot'.
xlim,ylim	Explicitly set the plot limits.
axes	Logical value indicating whether both axes should be drawn on the plot.
ann	Annotate the plots with axis titles and overall titles.
xlab,ylab	Labels for the X and Y axes.
x_axis_labels	Labels for the violins.
main	Title for the violin plot.
col	Fill color for the violin(s). Will be recycled.
median_col	Fill color for the median mark.
show_mean	Whether to plot the mean as well as the median.
mean_pch	Symbol to use for the mean.
mean_pch_col	Fill color for the mean symbol.
...	Extra arguments passed to 'polygon' used for representing violin(s).

Details

'violin_plot' displays one or more violin plots by drawing rotated kernel density curves on each side of box plots.

Value

nil

Author(s)

Darshan Baral

Examples

```
# plotting a data frame
violin_plot(mtcars)

set.seed(42)
normvar<-c(rnorm(49),-3)
unifvar<-runif(50,-2,2)
normvar2<-rnorm(45)

# plotting a matrix
violin_plot(matrix(c(normvar,unifvar),ncol=2),
 main="Default Plot",x_axis_labels=c("Normal","Uniform"))

# plotting with different colors and with at specified
violin_plot(matrix(c(normvar,unifvar),ncol=2),at=1:3,
 main="Different colors and extra space",
 x_axis_labels=c("Normal","Uniform","Normal"),
 show_outliers=TRUE,col=c("blue","red"),median_col="lightgray",
 pch=6)

# adding a violin to existing plot
violin_plot(normvar2,at=3,add=TRUE,col="green",violin_width=1)
```

---

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=NA,ylab=NULL,xaxis=TRUE,...)
```

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.
xaxis	Whether to display an X axis.
...	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.

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'.

Author(s)

Jim Lemon and Hadley Wickham - thanks to Ben Graf for asking for a custom x axis option and Martin Maechler for fixing the barplot problem

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,titlepos=NA,...)
```

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.
`titlepos`	The horizontal position of the title in user units of the zoomed plot.
`...`	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.

It just has. If '`rxlim`' is set to NA, '`locator`' will be called and the user can define the zoomed rectangle by clicking on each corner. This is a shameless ripoff of a suggestion by Greg Snow on the help list. Thanks, Greg.

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",titlepos=-1.5)
```

Index