The plotrix Package

September 23, 2007

Version 2.2-7

Date 2007-09-18

Title Various plotting functions

Author Jim Lemon <jim@bitwrit.com.au>, Ben Bolker <bolker@zoo.ufl.edu>,

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

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

License GPL Version 2 or later.

R topics documented:

arctext ....................................................... 3
axis.break ................................................... 4
axis.mult ..................................................... 5
barp ............................................................ 6
bin.wind.records ............................................ 8
boxed.labels ............................................... 9
brkdn.plot .................................................. 10
centipede.plot ............................................. 12
clean.args ................................................. 14
clock24.plot ................................................. 15
cluster.overplot .......................................... 16
color.gradient .............................................. 17
color.id .................................................... 18
color.legend ............................................... 18
color.scale .................................................. 20
color.scale.lines ......................................... 21
color2D.matplot ........................................... 22
corner.loc ............................................... 24
count.overplot ............................................ 25
cylindrect .................................................. 26
dispbars ..................................................... 27
R topics documented:

draw.arc ................................................................. 28
draw.circle .............................................................. 29
draw.tilted.sector ..................................................... 30
emptyspace ............................................................... 31
fan.plot ................................................................. 32
feather.plot ............................................................. 33
floating.pie ............................................................. 35
gantt.chart ............................................................. 36
gap.barplot ............................................................ 38
gap.boxplot ............................................................ 39
gap.plot ................................................................. 41
gget.breaks .............................................................. 42
gget.gantt.info .......................................................... 43
gget.segs ................................................................. 44
gget.soil.texture ...................................................... 45
gget.triprop ............................................................ 46
gradient.rect .......................................................... 47
hexagon ................................................................. 48
multistat ............................................................... 49
multisymbolbox ....................................................... 50
oz.windrose ............................................................ 51
oz.windrose.legend .................................................... 52
pie.labels ............................................................... 53
pie3D ................................................................. 54
pie3D.labels ........................................................... 55
plotCI ................................................................. 57
polar.plot .............................................................. 58
polygon.shadow ...................................................... 59
pyramid.plot .......................................................... 61
radial.plot .............................................................. 62
rescale ................................................................. 64
sizeplot ................................................................. 65
smoothColors .......................................................... 66
soil.texture ............................................................. 67
soils ................................................................. 69
setLabels ................................................................. 69
staxlab ................................................................. 70
std.error ................................................................. 71
symbolbarplot .......................................................... 72
symbolbox ............................................................. 73
taylor.plot .............................................................. 74
textbox ................................................................. 76
thigmophobe ............................................................ 77
thigmophobe.labels .................................................... 78
triax.abline ............................................................. 79
triax.frame .............................................................. 80
triax.plot ............................................................... 82
triax.points ............................................................ 84
arctext

Display text on a circular arc.

Description

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

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

Usage

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

Arguments

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

Value

nil

Author(s)

Jim Lemon - Thanks to Suhas Parandekar for the idea.

See Also

text
axis.break

Place a "break" mark on an axis

Description

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

Usage

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

Arguments

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

Value

nil

Note

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

Author(s)

Jim Lemon and Ben Bolker
See Also

gap.plot

Examples

plot(3:10,main="Axis break test")
# put a break at the default axis and position
axis.break()
axis.break(2,2.9,style="zigzag")
if(dev.interactive()) par(ask=TRUE)
twogr<-c(rnorm(10)+4,rnorm(10)+20)
gap.plot(twogr,gap=c(8,16),xlab="Index",ylab="Group values",
main="Two separated groups with gap axis break")
par(ask=FALSE)

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
at
labels
mult
mult.label
mult.line
mult.labelpos

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

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

barp

A bar plotting routine

Description

Display a bar plot

Usage

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

Arguments

- **height**: A numeric vector, matrix or data frame that will be represented as the heights of bars.
- **width**: Half the width of the bars in X axis units.
- **names.arg**: The labels for the bars or groups of bars.
- **legend.lab**: Labels for an optional legend. If NULL, no legend is displayed.
- **legend.pos**: Optional position for the legend as a list with x and y components. The default is to call `emptyspace` to position the legend. If this is NA, `locator` will be called.
barp

col  The fill colors for the bars. The default is no fill.
border  The border for the bars.
main  The title at the top of the plot.
xlab,ylab  The labels for the X and Y axes respectively.
xlim,ylim  Optional horizontal and vertical limits for the plot.
staxx,staxy  Whether to use staxlab to stagger the X or Y axis tick labels.
height.at  Optional positions of the tick marks on the Y axis.
height.lab  Optional tick labels for the Y axis.
cex.axis  Character expansion for the axis labels.
cylindrical  Whether to give the bars a cylindrical appearance by shading them.
shadow  Whether to place a shadow behind the bars.

details

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

and the value from table can also be passed as x.

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 treated as usual if the length of col is not equal to that of height. If height is a matrix or data frame, the user may pass a vector of colors equal to the number of rows in height or a matrix of colors of the same dimensions as height. Other sequences of color will probably not result in an easy to interpret plot.

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

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

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

value

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

x  The centers of the bars displayed.
y  The heights of the bars.
Classify wind direction and speed records.

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Jim Lemon

See Also

oz.windrose

Examples

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

boxed.labels Place labels in boxes

Description

Places labels in boxes on an existing plot

Usage

boxed.labels(x,y=NA,labels,bg="white",border=TRUE,xpad=0.6,ypad=0.6,...)
brkdn.plot

A point/line plotting routine

Description

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

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.
- `labels` Text strings
- `border` Whether to draw borders around the rectangles.
- `xpad, ypad` Half the value of the proportion of the rectangles to the extent of the text within.
- `...` additional arguments passed to `text`.

Value

`nil`

Note

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

Author(s)

Jim Lemon

See Also

- `spread.labels`
- `thigmophobe.labels`

Examples

```r
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)
readline("Press <Enter> to continue")
# now label a barplot
xpos <- barplot(c(1, 3, 2, 4))
boxed.labels(xpos, 0.5, nums[1:4])
```
**Usage**

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

**Arguments**

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

Value

A list of two matrices of dimension \( \text{length(levels(b)) by length(levels(c))} \). The first contains the measures of central tendency calculated and its name is the name of the function passed as \text{mct}. The second contains the measures of dispersion and its name is the name of the function passed as \text{md}.

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

Author(s)

Jim Lemon

See Also

dispbars

Examples

test.df<-data.frame(a=rnorm(80)+4,b=rnorm(80)+4,c=rep(LETTERS[1:4],each=20),
d=rep(rep(letters[1:4],each=4),5))
# first use the default values
brkdn.plot("a","c","d",test.df,pch=1:4,col=1:4)
# now jazz it up a bit using medians and median absolute deviations
# and some enhancements
bp<-brkdn.plot("a","c","d",test.df,main="Test of the breakdown plot",
mct="median",md="mad",xlab="Temperature range", ylab="Cognition",
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)

centipede.plot  Display a centipede plot

Description

Displays a centipede plot on the current graphics device.

Usage

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

centipede.plot

Display a centipede plot

Description

Displays a centipede plot on the current graphics device.

Usage

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

centipede.plot
centipede.plot 13

Arguments

segs  a matrix of midpoints and limits calculated by `get.segs` OR a `dstat` object returned by `brkdn`.
mct  The function to use in calculating the midpoint of each segment.
lower.limit  The functions to use in calculating the lower limits for each subset of the data.
upper.limit  The functions to use in calculating the upper limits.
left.labels  The variable or subset labels to place at the left margin of the plot. Default values are provided.
right.labels  The variable or subset labels to place at the right margin of the plot.
sort.segs  Whether to sort the segments in ascending order.
main  Optional title for the plot.
xlab  Optional x axis label for the plot. The default NA displays a text label showing the midpoint and limit functions.
vgrid  Optional vertical line(s) to display on the plot.
mar  Margin widths for the plot. Defaults to c(4,5,1,4) or c(4,5,3,4) if there is a title.
col  The color(s) of the limit lines and borders of the midpoint markers.
bg  The color(s) to fill the midpoint markers.
...  additional arguments passed to `plot`.

Details

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

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

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

Value

nil.

Author(s)

Jim Lemon

See Also

get.segs
Examples

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

---

clean.args

Remove inappropriate arguments from an argument list

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

Usage

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

Arguments

argstr a named list of arguments, e.g. from

list{...}

fn a function

exclude.repeats (logical) remove repeated arguments?

exclude.other a character vector of names of additional arguments to remove

dots.ok should "..." be allowed in the argument list?

Value

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

Author(s)

Ben Bolker

Examples

tststr <- list(n=2,mean=0,sd=1,foo=4,bar=6)
clean.args(tststr,rnorm)
try(do.call("rnorm",tststr))
do.call("rnorm",clean.args(tststr,rnorm))
remove.args(tststr,rnorm)
## add example of combining arg. lists?
clock24.plot  
**Plot values on a 24 hour "clockface".**

Description

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

Usage

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

Arguments

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

Value

nil

Author(s)

Jim Lemon

See Also

polar.plot, radial.plot

Examples

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

Usage

cluster.overplot(x,y,away=NA,tol=NA)

Arguments

x, y Numeric data vectors or the first two columns of a matrix or data frame. Typically the x/y coordinates of points to be plotted.

away How far to move overlying points in user units. Defaults to the width of a lower case "o" in the x direction and 5/8 of the height of a lower case "o" in the y direction.

tol The largest distance between points that will be considered to be overlying. Defaults to 1/2 of the width of a lower case "o" in the x direction and 1/2 of the height of a lower case "o" in the y direction.

Value

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

Author(s)

Jim Lemon

See Also

count.overplot, sizeplot

Examples

xy.mat<-cbind(sample(1:10,200,TRUE),sample(1:10,200,TRUE))
plot(cluster.overplot(xy.mat),main="Cluster overplot test")
color.gradient  

**Calculate an arbitrary sequence of colors.**

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

A vector of hexadecimal color values as used by col.

**Note**

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

**Author(s)**

Jim Lemon

**See Also**

rescale, approx.color.scale

**Examples**

# try it with red and blue endpoints and green midpoints.
color.gradient (c(0,1), c(1,0.6,0.4,0.3,0), c(0.1,0.6))
color.id  
Identify closest match to a color

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

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

Arguments
```
col  a color specified as a hex string
```

Details
finds the color with the minimum squared distance in RGB space

Value
the name of the closest match

Author(s)
Ben Bolker

See Also
col2rgb, colors

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

color.legend  
Legend matching categories or values to colors

Description
Display a color legend on a plot

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

color.legend

Arguments

xl,yb,xr,yt  The lower left and upper right coordinates of the rectangle of colors in user coordinates.

legend   The labels that will appear next to some or all of the colors.

rect.col   The colors that will fill the rectangle.

cex   Character expansion factor for the labels.

align   How to align the labels relative to the color rectangle.

gradient   Whether to have a horizontal (x) or vertical (y) color gradient.

...   Additional arguments passed to text.

Details

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

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

Value
nil

Author(s)
Jim Lemon

See Also
color.gradient, gradient.rect

Examples

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

*Turn values into colors.*

**Description**

Transform numeric values into colors.

**Usage**

```
color.scale(x, redrange=NA, greenrange=NA, bluerange=NA, extremes=NA)
```

**Arguments**

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

**Details**

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

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

**Value**

A vector or matrix of hexadecimal color values.

**Note**

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

**Author(s)**

Jim Lemon

**See Also**

`rescale`, `col2rgb`
color.scale.lines

Examples

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

color.scale.lines  Line segments with scaled colors.

Description

Display line segments with colors scaled to numeric values.

Usage

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

Arguments

x, y  Numeric vectors or a list with at least two components, the first two of which must be named x and y.

reds, greens, blues  Color ranges into which to scale the numeric values.

col  One or more colors to use for the resultant lines. Will be recycled if necessary.

colvar  A numeric vector from which to scale the colors.

...  Additional arguments passed to segments.

Details

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

Value

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

Author(s)
Jim Lemon

See Also
color.scale

Examples

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

color2D.matplot  

Display a numeric matrix as color matrix

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

Usage

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

Arguments

- `x`  
  data values
- `redrange`, `greenrange`, `bluerange`  
  the ranges of red, green and blue that will be scaled to represent the range of numeric values
- `extremes`  
  The colors for the extreme values of `x`. Takes precedence over the color ranges.
- `show.legend`  
  whether to display a color legend with the extreme numeric values in the lower left corner of the plot. If the default is not suitable, call `color.legend` separately.
- `nslices`  
  The number of color "slices" in the legend.
color2D.matplot

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.
vcol       The color for the value display.
vceX       The character expansion for the value display.
...        arguments passed to plot.

Details

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

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

Value

nil

Note

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

Author(s)

Jim Lemon (thanks to Ashoka Polpitiya for axes)

See Also

color.scale, image

Examples

```r
x <- matrix(rnorm(1024) + sin(seq(0, 2 * pi, length=1024)), nrow=32)
if (dev.interactive()) par(ask=TRUE)
color2D.matplot(x, c(1, 0), c(0, 0), c(0, 1), show.legend=TRUE,
    xlab="Columns", ylab="Rows", main="2D matrix plot")
# now do hexagons
color2D.matplot(x, c(1, 0), c(0, 0), c(0, 1), show.legend=TRUE,
    xlab="Columns", ylab="Rows", do.hex=TRUE, main="2D matrix plot (hexagons)")
# 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)
```
corner.loc

Find corner locations and display a label

Description

Finds the coordinates in user parameters of a specified corner of the figure region and/or puts a label there

Usage

```r
corner.loc(x=-1,y=1,xoff=0.05,yoff=0.05)
corner.label(label=NULL,x=-1,y=1,xoff=0.05,yoff=0.05,...)
```

Arguments

- `label`: text to plot in the appropriate corner
- `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`: horizontal offset as a proportion of the total plot size
- `yoff`: vertical offset as a proportion of the total plot size
- `...`: further arguments to the `text` command for the label

Value

`corner.loc` returns a list of the `x` and `y` positions of the corner

Author(s)

Ben Bolker

Examples

```r
plot(1:10,1:10)
corner.label("A")
corner.loc(1,1)
corner.label("B",y=-1,x=1)
```
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,...)

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.

...  
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 test",
xlab="X values",ylab="Y values")
cylindrect  

Display an apparent cylinder

Description

Display rectangles shaded to appear like cylinders.

Usage

\[
\text{cylindrect}(xleft, ybottom, xright, ytop, col, gradient="x", nslices=50)
\]

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.
- **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

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

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

Value

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

Author(s)

Jim Lemon

See Also

\text{gradient.rect}
Examples

```r
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")
```

=='dispbars'==' Dispersion bars

Description

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

Usage

```r
dispbars(x,y,ulim,llim=ulim,arrow.cap=0.02,arrow.gap=NA,...)
```

Arguments

- `x, y` x and y position of the centers of the bars
- `ulim, llim` The extent of the dispersion measures.
- `arrow.cap` The width of the cap at the outer end of each bar as a proportion of the width of the plot.
- `arrow.gap` The gap to leave at the inner end of each bar. Defaults to two thirds of the height of a capital "O".
- `...` additional arguments passed to `arrows`.

Details

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`
**draw.arc**

**Description**

Draw one or more arcs using classic graphics.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

Returns a matrix of expanded arguments invisibly.

**Author(s)**

Gabor Grothendieck

**Examples**

```r
plot(1:10, asp = 1)
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.circle

Draw a circle.

Description

Draws a circle on an existing plot.

Usage

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

Arguments

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

Value

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

Note

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

Author(s)

Jim Lemon

See Also

`polygon`

Examples

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

Description

Displays a 3D pie sector.

Usage

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

Arguments

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

Details

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

Value

The bisector of the pie sector in radians.

Author(s)

Jim Lemon

See Also

`pie3D`
Find an empty space on a plot.

**Description**

Try to find the largest empty rectangle on a plot.

**Usage**

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

**Arguments**

- `x`, `y`  
  x and y positions of the points or centers and heights of the bars

- `bars`  
  Whether to add x and y points to represent the bars if the plot is a barplot.

**Details**

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

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

**Value**

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

**Author(s)**

Jim Lemon

**Examples**

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

Display a fan plot.

Description

Displays numerical values as the arcs of overlapping sectors.

Usage

fan.plot(x, edges=200, radius=1, col=NULL, align.at=NULL, labels=NULL, labelpos=NULL, label.radius=1.1, 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).
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 on the circumference of the complete circle. Default is no ticks, TRUE gives the number of ticks equal to the integer sum of x, which is fairly sensible if x is a vector of integers.
include.sumx
  Whether to include the sum of all x values as the largest sector.
...
  Additional arguments passed to polygon.

Details

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

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

Display vectors along a horizontal reference line.

Description

Displays vectors along a line usually representing time or position.

Usage

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

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

Details

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

Value

`nil`

Author(s)

Jim Lemon, Eduardo Klein

See Also

`spread.labels`

Examples

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

Display a floating pie chart

Description

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

Usage

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

Arguments

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

Value

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

Note

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

Author(s)

Jim Lemon

See Also

pie.labels, boxed.labels, polygon.shadow
Examples

```r
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(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"))

```

gantt.chart

Display a Gantt chart

Description

Displays a Gantt chart with priority coloring

Usage

```r
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="", ylab="")
```

Arguments

- `x` a list of task labels, start/end times and task priorities as returned by `get.gantt.info`. If this is not present, `get.gantt.info` will be called.
- `format` the format to be used in entering dates/times (see `strptime`).
- `xlim` the horizontal limits of the plot.
- `taskcolors` a vector of colors used to illustrate task priority.
- `priority.legend` whether to display a priority color legend.
- `vgridpos` optional positions of the vertical grid lines.
- `vgridlab` optional labels for the vertical grid lines.
- `vgrid.format` format for the vertical grid labels.
- `half.height` the proportion of the spacing between task bars that will be filled by the bar on each side - 0.5 will leave no space.
gantt.chart

hgrid logical - whether to display grid lines between the bars.
main the title of the plot - note that this is actually displayed using mtext.
ylab vertical axis label - usually suppressed.

Details

If task priority colors are not wanted, simply mark every task as priority 1 or set taskcolors to a single value to suppress the coloring.

Value

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

Author(s)

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

See Also

get.gantt.info

Examples

Ymd.format<~"%Y/%m/%d"
gantt.info<-list(labels=
c("First task","Second task","Third task","Fourth task","Fifth task"),starts=as.POSIXct(strptime(c("2004/01/01","2004/02/02","2004/03/03","2004/05/05","2004/09/09"),format=Ymd.format)),ends=as.POSIXct(strptime(c("2004/03/03","2004/05/05","2004/05/05","2004/08/08","2004/12/12"),format=Ymd.format)),priorities=c(1,2,3,4,5))
vgridpos<as.POSIXct(strptime(c("2004/01/01","2004/02/01","2004/03/01","2004/04/01","2004/05/01","2004/06/01","2004/07/01","2004/08/01","2004/09/01","2004/10/01","2004/11/01","2004/12/01"),format=Ymd.format))
vgridlab<c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec")
gantt.chart(gantt.info,main="Calendar date Gantt chart (2004)",priority.legend=TRUE,vgridpos=vgridpos,vgridlab=vgridlab,hgrid=TRUE)
Display a barplot with a gap (missing range) on one axis

Description

Displays a barplot with a missing range.

Usage

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

Arguments

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

Details

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

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

Value

The center positions of the bars.

Author(s)

Jim Lemon
gap.boxplot

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,plot=TRUE,border=par("fg"),col=NULL,log="",
            pars=list(boxwex=0.8,staplewex=0.5,outwex=0.5),
            horizontal=FALSE,add=FALSE,at=NULL,main=NULL)

Arguments

  x              numeric vector or a list of vectors

  ...            arguments passed to boxplot.

  gap            the range(s) to be omitted - a list with two components, top and bottom each
                  specifying a range to omit. The default range of c(NA, NA) means no omitted
                  range

  range          how far to extend the whiskers, (see boxplot)

  width          the relative widths of the boxes

  varwidth       if TRUE, box widths are proportional to the square roots of the number of ob-
                  servations

  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

  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
gap.boxplot

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
pars        optional parameters for consistency with boxplot
horizontal  whether to plot horizontal boxplots - currently does nothing
add         whether to add the boxplot(s) to a current plot - currently does nothing
at          optional horizontal locations for the boxplots - currently does nothing
main        a title for the plot

Details

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

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

Value

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

Author(s)

Jim Lemon

See Also

gap.barplot,gap.plot

Examples

twovec<-list(vec1=rnorm(30),-6),vec2=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**

```r
gap.plot(x, y, gap, gap.axis = "y", xticlab, xtics = NA, yticlab, ytics = NA, 
col = rep(par("col"), length(x)), xlim, ylim, pch = rep(1, length(x)), ...)
```

**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
- `xticlab` labels for the x axis ticks
- `xtics` position of the x axis ticks
- `yticlab` labels for the y axis ticks
- `ytic` position of the y axis ticks
- `col` color(s) in which to plot the values
- `xlim, ylim` the plot limits.
- `pch` symbols to use in plotting.
- `...` arguments passed to `plot`.

**Details**

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

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.

**Value**

nil

**Author(s)**

Jim Lemon and Ben Bolker
get.breaks

See Also

gap.barplot

Examples

twogrpg <- c(rnorm(5)+4, rnorm(5)+20, rnorm(5)+5, rnorm(5)+22)
gpcol <- c(2,2,2,2,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,5)
gap.plot(twogrpg, gap=c(8,16), xlab="Index", ylab="Group values",
        main="Plot gap on Y axis", col=gpcol)
gap.plot(twogrpg, rnorm(20), gap=c(8,16), gap.axis="x", xlab="X values",
        xtics=c(4,7,17,20), ylab="Y values", main="Plot gap on X axis")
threegrp <- c(twogrpg, rnorm(10)+40)
gap.plot(threegrp, gap=c(8,16,25,35),
        xlab="X values", ylab="Y values",
        main="Test two gap plot",
        ytics=c(4,6,18,20,22,38,40),
        pch=c(rep(2,10), rep(3,10), rep(4,10)),
        col=c(rep(2,10), rep(3,10), rep(4,10)))

gap.barplot

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
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

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

### Arguments

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

### Details

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

### Value

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

### Author(s)

Jim Lemon

### See Also

`centipede.plot`
get.soil.texture Enter soil texture data.

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Sander Oom and Jim Lemon

See Also

soil.texture, get.triprop

Examples

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

Enter three proportion data - usually soil textures.

Description

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

Usage

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

Arguments

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

Details

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

Value

A matrix of the components of one or more observations.

Author(s)

Jim Lemon

See Also

triax.plot, soil.texture

Examples

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

```r
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**

```r
# 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`
**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, ...)  
```

**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
- `...`: additional arguments to `hist` or `barplot`

**Value**

plots a side-by-side barplot of the histograms

**Note**

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

**Author(s)**

Ben Bolker

**See Also**

`hist`, `barplot`

**Examples**

```r  
l <- list(runif(10)*10, 1:10, c(1,1,1,1,4,8))  
multhist(l)  
```
multsymbolbox  
Draw boxes filled with symbols

Description

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

Usage

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

Arguments

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

Value

```
none
```

Author(s)

Ben Bolker

Examples

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

Display an Australian wind rose.

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

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

Arguments

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

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

Value
nil

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

Author(s)
Jim Lemon (thanks to Anna in the Sydney BoM office)
See Also

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

Examples

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

Description

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

Usage

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

Arguments

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

Value

`nil`

Author(s)

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

See Also

oz.windrose

Examples

```r
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)
```
pie.labels  

Place labels on a pie chart

Description

Places labels on a pie chart

Usage

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

Arguments

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

Value

`nil`

Note

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

Author(s)

Jim Lemon

See Also

`floating.pie`, `boxed.labels`

Examples

```
pieval<-c(2,4,6,8)
pplot(1:5,type="n",axes=FALSE)
box()
bisect.angles<-floating.pie(3,3,pieval)
pie.labels(3,3,bisect.angles,c("two","four","six","eight"))
```
pie3D  
Display a 3D pie chart

Description
Displays a 3D pie chart with optional labels.

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

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

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

Description

Displays labels on a 3D pie chart.

Usage

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

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

Details

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

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

Value

- `nil`

Author(s)

Jim Lemon

See Also

- `pie3D`, `draw.tilted.sector`

Examples

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

Plot confidence intervals/error bars

Description

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

Usage

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

Arguments

x The x coordinates of points in the plot
y The y coordinates of points in the plot
uiw The width of the upper portion of the confidence region, or (if liw is missing)
     the width of both halves of the confidence region
liw The width of the lower portion of the confidence region (if missing, the function
     assumes symmetric confidence bounds)
ui The absolute upper limit of the confidence region
li The absolute lower limit of the confidence region
err The direction of error bars: "x" for horizontal, "y" for vertical ("xy" would be
     nice but is not implemented yet; don’t know quite how everything would be
     specified. See examples for composing a plot with simultaneous horizontal and
     vertical error bars)
gap Size of gap in error bars around points (default 0:gap=TRUE gives gap size of
     0.01)
sfrac Scaling factor for the size of the "serifs" (end bars) on the confidence bars, in
     x-axis units
add If FALSE (default), create a new plot; if TRUE, add error bars to an existing
     plot.
slty Line type of error bars
scol Color of error bars: if col is specified in the optional arguments, scol is set
     the same; otherwise it’s set to par(scol)
pt.bg Background color of points (use pch=21, pt.bg=par("bg") to get open points
     superimposed on error bars)
...

Any other parameters to be passed through to plot.default, points, arrows,
etc. (e.g. lwd, col, pch, axes, xlim, ylim). xlim and ylim are set by
default to include all of the data points and error bars. xlab and ylab are set
to the names of x and y. If pch=-NA, no points are drawn (e.g. leaving room
for text labels instead)
Value

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

Author(s)

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

See Also

boxplot

Examples

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

---

**polar.plot**

Plot values on a circular grid of 0 to 360 degrees.

Description

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

Usage

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

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

Value
nil

Author(s)
Jim Lemon

See Also
radial.plot

Examples

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

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 the light at the upper left. This effect may also be used as a text background.

Value

- nil

Note

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

Author(s)

- Jim Lemon

See Also

- polygon

Examples

```r
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.shadows(c(1.2,2,2,1.2),c(4.8,4.8,4.8,4.8),col=c("#ffff00","#cccc00"))
polygon(c(1.2,2,2,1.2),c(4.8,4.8,4.8,4.8),col=c("#ff0000"))
# a green triangle on a light blue square with a big offset
polygon(c(4,5,5),c(2,2,1),col="#aaaaff")
polygon.shadows(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.shadows(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
as a circular
text background",cex=1.5)
```
Description

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

Usage

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

Arguments

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

Details

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

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.

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

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

Author(s)

Jim Lemon

See Also

`rect`

Examples

```r
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","80-84","85+")
xycol=color.gradient(c(0,0,0.5,1),c(0,0,0.5,1),c(1,1,0.5,1),18)
xxcol=color.gradient(c(1,1,0.5,1),c(0.5,0.5,0.5,1),c(0.5,0.5,0.5,1),18)
par(mar=pyramid.plot(xy.pop,xx.pop,labels=agelabels,
main="Australian population pyramid 2002",xycol=xycol,xxcol=xxcol))
```

---

**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

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

Arguments

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

labels
labels are 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.

start
Where to place the starting (zero) point. Defaults to the 3 o’clock position.

clockwise
Whether to interpret positive positions as clockwise from the starting point. The default is counterclockwise.

rp.type
Whether to draw (r)adial lines, a (p)olygon, (s)ymbols or some combination of these.

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.

mar
Margins for the plot. Allows the user to leave space for legends, etc.

show.grid
Logical - whether to draw a circular grid.

show.radial.grid
Whether to draw radial lines to the plot labels.

grid.col
Color of the circular grid.

grid.bg
Fill color of above.

grid.left
Whether to place the radial grid labels on the left side.

point.symbols
The symbols for plotting (as in pch).

point.col
Colors for the symbols.

show.centroid
Whether to display a centroid.

radial.lim
The range of the grid circle. Defaults to range(pretty(lengths)).

...
Additional arguments are passed to plot.

Details

radial.plot displays a plot of radial lines, polygon(s), symbols or a combination of these centered at the midpoint of the plot frame, the lengths, vertices or positions corresponding to the numeric magnitudes of the data values. If show.centroid is TRUE, an enlarged point at the centroid of values is displayed.

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

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

Value

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

Author(s)

Jim Lemon

Examples

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

sizeplot

Plot with repeated symbols by size

Description

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

Usage

```r
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 \texttt{add=TRUE}.

Author(s)

Ben Bolker

See Also

\texttt{symbols}

Examples

\begin{verbatim}
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)
\end{verbatim}

\texttt{smoothColors} \hspace{1cm} \textit{Build a vector of color values.}

Description

\texttt{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

\texttt{smoothColors(..., alpha=NA)}

Arguments

\begin{verbatim}
... an arbitrary sequence of color names and integers beginning and ending with a color name.
alpha optional alpha (transparency) value.
\end{verbatim}

Value

A vector of hexadecimal color values as used by \texttt{col}.

Author(s)

Barry Rowlingson
soil.texture

See Also
color.gradient.rgb

Examples

```r
plot(1:10,main="Test opaque colors",type="n",axes=FALSE)
box()
```

---

soil.texture

Soil texture triangle plot

Description

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

Usage

```r
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.
soil.texture

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.
ltb.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 soil.texture 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


See Also

gt.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)
soil.texture(soils[1:6,], main="GRID AND LEGEND",
      show.grid=TRUE, pch=4, col.symbols=1:6, show.legend=TRUE)

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

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,offset,bg="white",border=FALSE,...)

Arguments
x, y  x and y data values
labels  text strings
offset  How far away from the data points to place the labels. Defaults to one quarter of the plot span.
bg  The color of the background rectangles on which to place the labels.
border  Whether to draw a border around the labels.
...  additional arguments passed to boxed.labels.
Value

nil

Note

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.

Author(s)

Jim Lemon

See Also

boxed.labels

Examples

x <- rnorm(10)/10
y <- sort(rnorm(10))
plot(x, y, xlim = c(-1, 1), type = "p")
ums <- c("one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten")
spread.labels(x, y, nums, 0.5)

staxlab

Place staggered labels on an axis

Description

Places labels on an axis in a regular staggered order

Usage

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

Arguments

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

nil

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

Author(s)

Jim Lemon

See Also

mtext

Examples

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

Description

Calculate 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.
symbolbarplot

Author(s)
Jim Lemon

See Also
sd

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?
symbolbox

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 multisymbolbox

Value
Nil

Note
This is a mostly a hack of barplot()

Author(s)
Ben Bolker

Examples

```r
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**

```r
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
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")

taylor.diagram

Taylor diagram

Description

Display a Taylor diagram.

Usage

taylor.diagram(ref,model,add=FALSE,col="red",pch=19,pos.cor=TRUE, xlab="",ylab="",main="Taylor Diagram",show.gamma=TRUE,ref.sd=FALSE, pcex=1,normalize=FALSE,...)
Arguments

- **ref**: numeric vector - the reference values.
- **model**: numeric vector - the predicted model values.
- **add**: whether to draw the diagram or just add a point.
- **col**: the color for the points displayed.
- **pch**: the type of point to display.
- **pos.cor**: whether to display only positive (TRUE) or all values of correlation (FALSE).
- **xlab, ylab**: plot axis labels.
- **main**: title for the plot.
- **show.gamma**: whether to display standard deviation arcs around the reference point (only for pos.cor=TRUE).
- **ref.sd**: whether to display the arc representing the reference standard deviation.
- **pcex**: character expansion for the plotted points.
- **normalize**: whether to normalize the models so that the reference has a standard deviation of 1.
- **...**: Additional arguments passed to plot.

Details

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

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

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

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

Examples

```r
# 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")
```

Description

Add text to plot, justified, in a box

Usage

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

Arguments

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

Details

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

**Value**

nil

**Author(s)**

Ben Bolker

**Examples**

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

---

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

**Description**

Find the direction away from the closest point

**Usage**

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

**Arguments**

- **x, y**  
  Numeric data vectors. Typically the x/y coordinates of plotted points.

**Details**

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

**Value**

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

**Note**

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

**Author(s)**

Jim Lemon
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.

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

Author(s)
  Jim Lemon
See Also

thigmophobe, text

Examples

```r
x <- rnorm(20)
y <- rnorm(20)
xlim <- range(x)
xspace <- (xlim[2] - xlim[1]) / 20
ylim <- range(y)
yspace <- (ylim[2] - ylim[1]) / 20
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
thigmophobe.labels(x, y, plotlabels, col=c(2:6, 8:12))
```

---

**triax.abline**

*Lines for triangle plot*

**Description**

Display lines on a triangle plot.

**Usage**

```r
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.
triax.frame

Triangle plot frame

Description

Display a three axis frame with optional grid.

Usage

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

Arguments

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

Details

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

Value

nil

Author(s)

Jim Lemon

See Also

triax.points, triax.abline

Examples

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

display a triangle plot with optional grid.

Usage

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

Arguments

- `x`: Matrix where each row is three proportions or percentages that must sum to 1 or 100 respectively.
- `main`: The title of the soil texture 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 thigmophobia.labels to label the points.
- `point.labels`: Optional labels for the points and/or legend.
- `col.symbols`: Color of the symbols representing each value.
- `pch`: Symbols to use in plotting values.
- `no.add`: Whether to restore the previous plotting parameters (TRUE) or leave them, allowing more points to be added.
- `...`: Additional arguments passed to points.
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 accommodate 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

```r
data(soils)
triax.plot(soils[1:10,],main="DEFAULT")
if(dev.interactive()) par(ask=TRUE)
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)
main.title<-
triax.plot(soils[1:6,],main="GRID AND LEGEND",show.grid=TRUE,
show.legend=TRUE,col.symbols=1:6,pch=4)
if(dev.interactive()) par(ask=FALSE)
```
triax.points  

Triangle plot points

Description
Display points on a triangle plot.

Usage
```r
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

```r
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)
```

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

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

Arguments

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

Details

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

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

A list containing:

- `breaks` - The endpoints of the ranges
- `counts` - The weighted counts
- `density` - The weighted counts divided by their sum.
- `mids` - The midpoints of the ranges and the bars displayed.
- `xname` - the name of x.
- `equidist` - Whether all ranges are equal to less than 1000th of the total range.

Author(s)

Jim Lemon and Hadley Wickham

See Also

`hist`

Examples

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

∗Topic aplot
  corner.loc, 22
  multisymbolbox, 48
  symbolbox, 71
  textbox, 74

∗Topic color
  color.id, 16

∗Topic hplot
  multhist, 47
  plotCI, 55
  sizeplot, 63
  symbolbarplot, 70

∗Topic misc
  arctext, 1
  axis.break, 2
  axis.mult, 4
  barp, 5
  bin.wind.records, 7
  boxed.labels, 8
  brcdn.plot, 9
  centipede.plot, 11
  clock24.plot, 13
  cluster.overplot, 14
  color.gradient, 15
  color.legend, 17
  color.scale, 18
  color.scale.lines, 19
  color2D.matplot, 20
  count.overplot, 23
  cylindrect, 24
  dispbars, 25
  draw.arc, 26
  draw.circle, 27
  draw.tilted.sector, 28
  empty.space, 29
  fan.plot, 30
  feather.plot, 31
  floating.pie, 33
  gantt.chart, 34
  gap.barplot, 36
  gap.boxplot, 37
  gap.plot, 39
  get.breaks, 40
  get.gantt.info, 41
  get.segs, 42
  get.soil.texture, 43
  get.triprop, 44
  gradient.rect, 45
  hexagon, 46
  oz.windrose, 49
  oz.windrose.legend, 50
  pie.labels, 51
  pie3D, 52
  pie3D.labels, 53
  polar.plot, 56
  polygon.shadow, 57
  pyramid.plot, 59
  radial.plot, 60
  rescale, 62
  smoothColors, 64
  soil.texture, 65
  soils, 67
  spread.labels, 67
  staxlab, 68
  std.error, 69
  taylor.diagram, 72
  thigmophobe, 75
  thigmophobe.labels, 76
  triax.abline, 77
  triax.frame, 78
  triax.plot, 80
  triax.points, 82
  weighted.hist, 83

∗Topic programming
  clean.args, 12
  approx, 16
  arctext, 1
  arrows, 25, 55
<table>
<thead>
<tr>
<th>axis</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis.break</td>
<td>2, 36, 38, 39</td>
</tr>
<tr>
<td>axis.mult</td>
<td>4</td>
</tr>
<tr>
<td>barp</td>
<td>5</td>
</tr>
<tr>
<td>barplot</td>
<td>6, 47, 83</td>
</tr>
<tr>
<td>bin.wind.records</td>
<td>7, 50</td>
</tr>
<tr>
<td>boxed.labels</td>
<td>8, 33, 51</td>
</tr>
<tr>
<td>boxplot</td>
<td>37, 38, 56</td>
</tr>
<tr>
<td>brkdn.plot</td>
<td>9</td>
</tr>
<tr>
<td>centipede.plot</td>
<td>11, 42</td>
</tr>
<tr>
<td>clean.args</td>
<td>12</td>
</tr>
<tr>
<td>clock24.plot</td>
<td>13, 62</td>
</tr>
<tr>
<td>cluster.overplot</td>
<td>14, 23</td>
</tr>
<tr>
<td>col2rgb</td>
<td>17, 19</td>
</tr>
<tr>
<td>color.gradient</td>
<td>15, 18, 65</td>
</tr>
<tr>
<td>color.id</td>
<td>16</td>
</tr>
<tr>
<td>color.legend</td>
<td>17</td>
</tr>
<tr>
<td>color.scale</td>
<td>16, 18, 20, 22</td>
</tr>
<tr>
<td>color.scale.lines</td>
<td>19</td>
</tr>
<tr>
<td>color2D.matplot</td>
<td>20, 46</td>
</tr>
<tr>
<td>colors</td>
<td>17</td>
</tr>
<tr>
<td>corner.label (corner.loc)</td>
<td>22</td>
</tr>
<tr>
<td>corner.loc</td>
<td>22</td>
</tr>
<tr>
<td>count.overplot</td>
<td>14, 15, 23</td>
</tr>
<tr>
<td>cylindrect</td>
<td>6, 24</td>
</tr>
<tr>
<td>dispbars</td>
<td>10, 25</td>
</tr>
<tr>
<td>draw.arc</td>
<td>26</td>
</tr>
<tr>
<td>draw.circle</td>
<td>27, 50</td>
</tr>
<tr>
<td>draw.tilted.sector</td>
<td>28, 53, 54</td>
</tr>
<tr>
<td>emptyspace</td>
<td>5, 29</td>
</tr>
<tr>
<td>fan.plot</td>
<td>30</td>
</tr>
<tr>
<td>feather.plot</td>
<td>31</td>
</tr>
<tr>
<td>floating.pie</td>
<td>31, 33, 51</td>
</tr>
<tr>
<td>gantt.chart</td>
<td>34, 41</td>
</tr>
<tr>
<td>gap.barplot</td>
<td>36, 37, 38, 40</td>
</tr>
<tr>
<td>gap.boxplot</td>
<td>37</td>
</tr>
<tr>
<td>gap.plot</td>
<td>3, 38, 39</td>
</tr>
<tr>
<td>get.breaks</td>
<td>40</td>
</tr>
<tr>
<td>get.gantt.info</td>
<td>34, 35, 41</td>
</tr>
<tr>
<td>get.segs</td>
<td>11, 12, 42</td>
</tr>
<tr>
<td>get.soil.texture</td>
<td>43, 66</td>
</tr>
<tr>
<td>get.triprop</td>
<td>43, 44</td>
</tr>
<tr>
<td>gradient.rect</td>
<td>6, 17, 18, 25, 45</td>
</tr>
<tr>
<td>hexagon</td>
<td>46</td>
</tr>
<tr>
<td>hist</td>
<td>41, 47, 83, 84</td>
</tr>
<tr>
<td>image</td>
<td>21, 22</td>
</tr>
<tr>
<td>mtext</td>
<td>5, 17, 69</td>
</tr>
<tr>
<td>multihist</td>
<td>47</td>
</tr>
<tr>
<td>multisymbolbox</td>
<td>48, 72</td>
</tr>
<tr>
<td>oz.windrose</td>
<td>8, 49, 50</td>
</tr>
<tr>
<td>oz.windrose.legend</td>
<td>50, 50</td>
</tr>
<tr>
<td>pie.labels</td>
<td>33, 51</td>
</tr>
<tr>
<td>pie3D</td>
<td>28, 52, 54</td>
</tr>
<tr>
<td>pie3D.labels</td>
<td>53, 53</td>
</tr>
<tr>
<td>plot.default</td>
<td>55</td>
</tr>
<tr>
<td>plotCI</td>
<td>55</td>
</tr>
<tr>
<td>points</td>
<td>55</td>
</tr>
<tr>
<td>polar.plot</td>
<td>14, 56, 62</td>
</tr>
<tr>
<td>polygon</td>
<td>27, 58</td>
</tr>
<tr>
<td>polygon.shadow</td>
<td>33, 57</td>
</tr>
<tr>
<td>pyramid.plot</td>
<td>59</td>
</tr>
<tr>
<td>radial.plot</td>
<td>14, 57, 60</td>
</tr>
<tr>
<td>rect</td>
<td>60</td>
</tr>
<tr>
<td>remove.args (clean.args)</td>
<td>12</td>
</tr>
<tr>
<td>rescale</td>
<td>16, 19, 62</td>
</tr>
<tr>
<td>rgb</td>
<td>65</td>
</tr>
<tr>
<td>sd</td>
<td>70</td>
</tr>
<tr>
<td>segments</td>
<td>25</td>
</tr>
<tr>
<td>sizeplot</td>
<td>14, 15, 23, 63</td>
</tr>
<tr>
<td>smoothColors</td>
<td>64</td>
</tr>
<tr>
<td>soil.texture</td>
<td>43, 44, 65</td>
</tr>
<tr>
<td>soils</td>
<td>67</td>
</tr>
<tr>
<td>spread.labels</td>
<td>9, 32, 67</td>
</tr>
<tr>
<td>staxlab</td>
<td>5, 6, 68</td>
</tr>
<tr>
<td>std.error</td>
<td>69</td>
</tr>
<tr>
<td>strptime</td>
<td>34, 41</td>
</tr>
<tr>
<td>symbolbarplot</td>
<td>70</td>
</tr>
<tr>
<td>symbolbox</td>
<td>71</td>
</tr>
<tr>
<td>symbols</td>
<td>64</td>
</tr>
<tr>
<td>taylor.diagram</td>
<td>72</td>
</tr>
<tr>
<td>text</td>
<td>2, 17, 76, 77</td>
</tr>
<tr>
<td>textbox</td>
<td>74</td>
</tr>
<tr>
<td>thigmophobe</td>
<td>75, 77</td>
</tr>
<tr>
<td>thigmophobe.labels</td>
<td>9, 66, 76, 76, 81, 82</td>
</tr>
<tr>
<td>triax.abline</td>
<td>77, 79, 81</td>
</tr>
</tbody>
</table>
INDEX

triax.frame, 78
triax.plot, 44, 66, 78, 80, 82
triax.points, 66, 79, 81, 82

weighted.hist, 83

xy.coords, 8