Package ‘gplots’

October 6, 2024

Title Various R Programming Tools for Plotting Data

Description Various R programming tools for plotting data, including:
- calculating and plotting locally smoothed summary function as ('bandplot', 'wapply'),
- enhanced versions of standard plots ('barplot2', 'boxplot2', 'heatmap.2', 'smartlegend'),
- manipulating colors ('col2hex', 'colorpanel', 'redgreen', 'greenred', 'bluered', 'redblue', 'rich.colors'),
- calculating and plotting two-dimensional data summaries ('ci2d', 'hist2d'),
- enhanced regression diagnostic plots ('lmplot2', 'residplot'),
- formula-enabled interface to 'stats::lowess' function ('lowess'),
- displaying textual data in plots ('textplot', 'sinkplot'),
- plotting dots whose size reflects the relative magnitude of the elements ('balloonplot', 'bubbleplot'),
- plotting ``Venn'' diagrams ('venn'),
- displaying Open-Office style plots ('ooplot'),
- plotting multiple data on same region, with separate axes ('overplot'),
- plotting means and confidence intervals ('plotCI', 'plotmeans'),
- spacing points in an x-y plot so they don't overlap ('space').

Depends R (>= 3.0)

Imports gtools, stats, caTools, KernSmooth, methods

Suggests grid, MASS, knitr, rmarkdown, r2d2, nlme

LazyData yes

VignetteBuilder knitr

Version 3.2.0

Date 2024-10-05

License GPL-2

URL https://github.com/talgalili/gplots,
https://talgalili.github.io/gplots/

BugReports https://github.com/talgalili/gplots/issues

NeedsCompilation no

Author Gregory R. Warnes [aut],
Ben Bolker [aut],
Lodewijk Bonebakker [aut],
Robert Gentleman [aut],
Wolfgang Huber [aut],
Andy Liaw [aut],
Thomas Lumley [aut],
Martin Maechler [aut],
Arni Magnusson [aut],
Steffen Moeller [aut],
Marc Schwartz [aut],
Bill Venables [aut],
Tal Galili [aut, cre]

Maintainer Tal Galili <`tal.galili@gmail.com`>

Repository CRAN

Date/Publication 2024-10-05 23:00:02 UTC

Contents

adjust_hsv 3
angleAxis 4
balloonplot 5
bandplot 9
barplot2 11
boxplot2 15
bubbleplot 17
catch.d 19
ci2d 20
col2hex 25
colorpanel 26
gplots-defunct 27
gplots-deprecated 29
heatmap.2 29
hist2d 39
lmplot2 41
lowess 43
ooplot.default 45
overplot 49
plotCI 51
plotmeans 53
qqnorm.aov 56
reorder.factor 57
residplot 58
rich.colors 59
rtPCR 60

sinkplot 62
space 63
textplot 65
venn 67
wapply 70

Index **73**

`adjust_hsv` *Adjust Color in HSV Space*

Description

Adjust color hue, saturation, and/or alpha value.

Usage

```
adjust_hsv(col, h=NULL, s=NULL, v=NULL, alpha=NULL)
```

Arguments

`col`	a color or vector of colors.
`h`	the desired hue.
`s`	the desired saturation.
`v`	the desired value.
`alpha`	the desired transparency.

Details

Colors can be specified as a color name, a hexadecimal string, or an integer.

Hue, saturation, value, and transparency are specified as values from 0 to 1, or `NULL` to leave unchanged.

Value

Adjusted colors in hexadecimal string format.

Author(s)

Arni Magnusson.

See Also

`col2rgb`, `rgb2hsv`, and `hsv` are the underlying functions used to convert and adjust the colors.

Examples

```
col <- "#123456"
col2 <- adjust_hsv(col, h=0.1)
col3 <- adjust_hsv(col, s=0.1)
col4 <- adjust_hsv(col, v=0.7)

barplot(rep(1, 4), col=c(col, col2, col3, col4))
```

`angleAxis` *Add a Axis to a Plot with Rotated Labels*

Description

Add a labeled axis to the current plot with rotated text

Usage

```
angleAxis(side, labels, at = 1:length(labels), srt = 45, adj, xpd = TRUE, ...)
```

Arguments

`side`	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
`labels`	character or expression vector of labels to be placed at the tickpoints.
`at`	the points at which tick-marks are to be drawn. Non-finite (infinite, NaN or NA) values are omitted.
`srt`	The string rotation in degrees. Defaults to 45 degrees (clockwise).
`adj`	Text justification. A value of 0 produces left-justified text, 0.5 centered text and 1 right-justified text. For `side=1` and `side=2`, the default value is `adj=1`. For `side=3` and `side=4` the default value is `adj=0`.
`xpd`	A logical value or NA. If FALSE, labels are clipped to the plot region, if TRUE, labels are clipped to the figure region, and if NA, labels are clipped to the device region.
`...`	optional arguments passed to `text`. Common examples are `col`, `cex`.

Details

This function augments the feature of the `axis` functon by allowing the axis labels to be rotated.

Author(s)

Gregory R. Warnes `<greg@warnes.net>`

See Also

`axis`

Examples

```
# create a vector with some values and long labels
values <- sample(1:10)
names(values) <- sapply(letters[1:10],
                        function(x) paste(rep(x, 10), sep="",collapse="")
                        )

# barplot labels are too long for the available space, hence some are not plotted
barplot(values)

# to add angled labels, tell barplot not to label the x axis, and store the bar location
at <- barplot(values, xaxt="n")
# then use angleAxs
angleAxis(1, at=at, labels = names(values))

# angle counter-clockwise instead
at <- barplot(values, xaxt="n")
angleAxis(1, at=at, labels = names(values), srt=-45, adj=0)

# put labels at the top
oldpar <- par()$mar
par(mar=c(1,4,5,2)+0.1)
at <- barplot(values, xaxt="n")
angleAxis(3, at=at, labels = names(values))
par(oldpar)

# put labels on the left
oldpar <- par()$mar
par(mar=c(5,5,3,2)+0.1)
at <- barplot(values, yaxt="n", horiz=TRUE)
angleAxis(2, at=at, labels = names(values))
par(oldpar)

# put labels on the right
oldpar <- par()$mar
par(mar=c(2,5,3,5)+0.1)
at <- barplot(values, yaxt="n", horiz=TRUE)
angleAxis(4, at=at, labels = names(values))
par(oldpar)

# specify colors for bars and labels
at <- barplot(values, xaxt="n", col=1:10)
angleAxis(1, at=at, labels = names(values), col=1:10)
```

`balloonplot` *Plot a graphical matrix where each cell contains a dot whose size reflects the relative magnitude of the corresponding component.*

Description

Plot a graphical matrix where each cell contains a dot whose size reflects the relative magnitude of the corresponding component.

Usage

```
balloonplot(x, ...)
## S3 method for class 'table'
balloonplot(x, xlab, ylab, zlab, show.zeros=FALSE,show.margins=TRUE,...)
## Default S3 method:
balloonplot(x,y,z,
                                xlab,
                                ylab,
                                zlab=deparse(substitute(z)),
                                dotsize=2/max(strwidth(19),strheight(19)),
                                dotchar=19,
                                dotcolor="skyblue",
                                text.size=1,
                                text.color=par("fg"),
                                main,
                                label=TRUE,
                                label.digits=2,
                                label.size=1,
                                label.color=par("fg"),
                                scale.method=c("volume","diameter"),
                                scale.range=c("absolute","relative"),
                                colsrt=par("srt"),
                                rowsrt=par("srt"),
                                colmar=1,
                                rowmar=2,
                                show.zeros=FALSE,
                                show.margins=TRUE,
                                cum.margins=TRUE,
                                sorted=TRUE,
                                label.lines=TRUE,
                                fun=function(x)sum(x,na.rm=T),
                                hide.duplicates=TRUE,
                                ... )
```

Arguments

x	A table object, or either a vector or a list of several categorical vectors containing grouping variables for the first (x) margin of the plotted matrix.
y	Vector or list of vectors for grouping variables for the second (y) dimension of the plotted matrix.
z	Vector of values for the size of the dots in the plotted matrix.
xlab	Text label for the x dimension. This will be displayed on the x axis and in the plot title.

`ylab`	Text label for the y dimension. This will be displayed on the y axis and in the plot title.
`zlab`	Text label for the dot size. This will be included in the plot title.
`dotsize`	Maximum dot size. You may need to adjust this value for different plot devices and layouts.
`dotchar`	Plotting symbol or character used for dots. See the help page for the points function for symbol codes.
`dotcolor`	Scalar or vector specifying the color(s) of the dots in the plot.
`text.size`, `text.color`	Character size and color for row and column headers
`main`	Plot title text.
`label`	Boolean flag indicating whether the actual value of the elements should be shown on the plot.
`label.digits`	Number of digits used in formatting value labels.
`label.size`, `label.color`	Character size and color for value labels.
`scale.method`	Method of scaling the sizes of the dot, either "volume" or "diameter". See below.
`scale.range`	Method for scaling original data to compute circle diameter. `scale.range="absolute"` scales the data relative to 0 (i.e, maps [0,max(z)] –> [0,1]), while `scale.range="relative"` scales the data relative to min(z) (i.e. maps [min(z), max(z)] –> [0,1]).
`rowsrt`, `colsrt`	Angle of rotation for row and column labels.
`rowmar`, `colmar`	Space allocated for row and column labels. Each unit is the width/height of one cell in the table.
`show.zeros`	boolean. If `FALSE`, entries containing zero will be left blank in the plotted matrix. If `TRUE`, zeros will be displayed.
`show.margins`	boolean. If `TRUE`, row and column sums are printed in the bottom and right margins, respectively.
`cum.margins`	boolean. If `TRUE`, marginal fractions are graphically presented in grey behind the row/column label area.
`sorted`	boolean. If `TRUE`, the rows will be arranged in sorted order by using the levels of the first y factor, then the second y factor, etc. The same process is used for the columns, based on the x factors
`label.lines`	boolean. If `TRUE`, borders will be drawn for row and column level headers.
`hide.duplicates`	boolean. If `TRUE`, column and row headers will omit duplicates within row/column to reduce clutter. Defaults to `TRUE`.
`fun`	function to be used to combine data elements with the same levels of the grouping variables `x` and `y`. Defaults to `sum`
`...`	Additional arguments passed to `balloonplot.default` or `plot`, as appropriate.

Details

This function plots a visual matrix. In each x,y cell a dot is plotted which reflects the relative size of the corresponding value of z. When `scale.method="volume"` the volume of the dot is proportional to the relative size of z. When `scale.method="diameter"`, the diameter of the dot is proportional to the the relative size of z. The "volume" method is default because the "diameter" method visually exaggerates differences.

Value

Nothing of interest.

Note

z is expected to be non-negative. The function will still operate correctly if there are negative values of z, but the corresponding dots will have 0 size and a warning will be generated.

Author(s)

Gregory R. Warnes <greg@warnes.net>

References

Function inspired by question posed on R-help by Ramon Alonso-Allende <allende@cnb.uam.es>.

See Also

plot.table.

bubbleplot provides an alternative interface and visual style based on scatterplots instead of tables.

Examples

```
# Create an Example Data Frame Containing Car x Color data
carnames <- c("bmw","renault","mercedes","seat")
carcolors <- c("red","white","silver","green")
datavals <- round(rnorm(16, mean=100, sd=60),1)
data <- data.frame(Car=rep(carnames,4),
                   Color=rep(carcolors, c(4,4,4,4) ),
                   Value=datavals )
# show the data
data

# generate balloon plot with default scaling
balloonplot( data$Car, data$Color, data$Value)


# show margin label rotation & space expansion, using some long labels
levels(data$Car) <- c("BMW: High End, German","Renault: Medium End, French",
 "Mercedes: High End, German", "Seat: Imaginary, Unknown Producer")

# generate balloon plot with default scaling
```

```
balloonplot( data$Car, data$Color, data$Value, colmar=3, colsrt=90)

# Create an example using table
xnames <- sample( letters[1:3], 50, replace=2)
ynames <- sample( 1:5, 50, replace=2)

tab <- table(xnames, ynames)

balloonplot(tab)

# Example of multiple classification variabls using the Titanic data
library(datasets)
data(Titanic)

dframe <- as.data.frame(Titanic) # convert to 1 entry per row format
attach(dframe)
balloonplot(x=Class, y=list(Survived, Age, Sex), z=Freq, sort=TRUE)

# colorize: surviors lightblue, non-survivors: grey
Colors <- Titanic
Colors[,,,"Yes"] <- "skyblue"
Colors[,,,"No"] <- "grey"
colors <- as.character(as.data.frame(Colors)$Freq)

balloonplot(x=list(Age,Sex),
            y=list(Class=Class,
                   Survived=reorder.factor(Survived,new.order=c(2,1))
                   ),
            z=Freq,
            zlab="Number of Passengers",
            sort=TRUE,
            dotcol = colors,
            show.zeros=TRUE,
            show.margins=TRUE)
```

---

bandplot *Plot x-y Points with Locally Smoothed Mean and Standard Deviation*

Description

Plot x-y points with curves for locally smoothed mean and standard deviation.

Usage

```
bandplot(x,...)
## S3 method for class 'formula'
bandplot(x, data, subset, na.action, ...,
            xlab=NULL, ylab=NULL, add = FALSE, sd = c(-2:2),
            sd.col=c("magenta", "blue", "red", "blue", "magenta"),
            sd.lwd=c(2, 2, 3, 2, 2),  sd.lty=c(2, 1, 1, 1, 2),
            method = "frac", width = 1/5, n=50)
## Default S3 method:
bandplot(x, y, ..., add = FALSE, sd = c(-2:2),
            sd.col=c("magenta", "blue", "red", "blue", "magenta"),
            sd.lwd=c(2, 2, 3, 2, 2),  sd.lty=c(2, 1, 1, 1, 2),
            method = "frac", width = 1/5, n=50)
```

Arguments

`x`	either formula providing a single dependent variable (y) and an single independent variable (x) to use as coordinates in the scatter plot or a numeric vector of x locations
`y`	numeric vector of y locations
`data`	an optional data.frame, list, or environment contianing the variables used in the model (and in `subset`). If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The factory-fresh default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
`...`	Additional plotting parameters
`xlab`, `ylab`	x and y axis labels
`add`	Boolean indicating whether the local mean and standard deviation lines should be added to an existing plot. Defaults to FALSE.
`sd`	Vector of multiples of the standard devation that should be plotted. `0` gives the mean, `-1` gives the mean minus one standard deviation, etc. Defaults to -2:2.
`sd.col`, `sd.lwd`, `sd.lty`	Color, line width, and line type of each plotted line.
`method`, `width`, `n`	Parameters controlling the smoothing. See the help page for `wapply` for details.

Details

`bandplot` was created to look for changes in the mean or variance of scatter plots, particularly plots of regression residuals.

The local mean and standard deviation are calculated by calling 'wapply'. By default, bandplot asks wapply to smooth using intervals that include the nearest 1/5 of the data. See the documentation of that function for details on the algorithm.

Value

Invisibly returns a list containing the x,y points plotted for each line.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

wapply, lowess

Examples

```
# fixed mean, changing variance
x <- 1:1000
y <- rnorm(1000, mean=1, sd=1 + x/1000 )
bandplot(x,y)
bandplot(y~x)

# fixed varance, changing mean
x <- 1:1000
y <- rnorm(1000, mean=x/1000, sd=1)
bandplot(x,y)

#
# changing mean and variance
#
x <- abs(rnorm(500))
y <- rnorm(500, mean=2*x, sd=2+2*x)

# the changing mean and dispersion are hard to see whith the points alone:
plot(x,y )

# regression picks up the mean trend, but not the change in variance
reg <- lm(y~x)
summary(reg)
abline(reg=reg, col="blue", lwd=2)

# using bandplot on the original data helps to show the mean and
# variance trend
bandplot(y ~ x)

# using bandplot on the residuals helps to see that regression removes
# the mean trend but leaves the trend in variability
bandplot(predict(reg),resid(reg))
```

---

barplot2 *Enhanced Bar Plots*

Description

An enhancement of the standard barplot() function. Creates a bar plot with vertical or horizontal bars. Can plot confidence intervals for each bar, a lined grid behind the bars, change plot area color and logarithmic axes may be used.

Usage

```
## Default S3 method:
barplot2(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, prcol = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
        plot.ci = FALSE, ci.l = NULL, ci.u = NULL,
        ci.color = "black", ci.lty = "solid", ci.lwd = 1, ci.width = 0.5,
        plot.grid = FALSE, grid.inc = NULL,
        grid.lty = "dotted", grid.lwd = 1, grid.col = "black",
        add = FALSE, panel.first = NULL, panel.last = NULL, ...)
```

Arguments

`height`	either a vector or matrix of values describing the bars which make up the plot. If `height` is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If `height` is a matrix and `beside` is `FALSE` then each bar of the plot corresponds to a column of `height`, with the values in the column giving the heights of stacked “sub-bars” making up the bar. If `height` is a matrix and `beside` is `TRUE`, then the values in each column are juxtaposed rather than stacked.
`width`	optional vector of bar widths. Re-cycled to length the number of bars drawn. Specifying a single value will no visible effect unless `xlim` is specified.
`space`	the amount of space (as a fraction of the average bar width) left before each bar. May be given as a single number or one number per bar. If `height` is a matrix and `beside` is `TRUE`, `space` may be specified by two numbers, where the first is the space between bars in the same group, and the second the space between the groups. If not given explicitly, it defaults to `c(0,1)` if `height` is a matrix and `beside` is `TRUE`, and to 0.2 otherwise.
`names.arg`	a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the `names` attribute of `height` if this is a vector, or the column names if it is a matrix.
`legend.text`	a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included. This is only useful when `height` is a matrix. In that case given legend labels should correspond to the rows of `height`; if `legend.text` is true, the row names of `height` will be used as labels if they are non-null.
`beside`	a logical value. If `FALSE`, the columns of `height` are portrayed as stacked bars, and if `TRUE` the columns are portrayed as juxtaposed bars.
`horiz`	a logical value. If `FALSE`, the bars are drawn vertically with the first bar to the left. If `TRUE`, the bars are drawn horizontally with the first at the bottom.

`density`	a vector giving the the density of shading lines, in lines per inch, for the bars or bar components. The default value of `NULL` means that no shading lines are drawn. Non-positive values of `density` also inhibit the drawing of shading lines.
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise), for the bars or bar components.
`col`	a vector of colors for the bars or bar components. By default, grey is used if `height` is a vector, and `heat.colors(nrow(height))` if `height` is a matrix.
`prcol`	the color to be used for the plot region.
`border`	the color to be used for the border of the bars.
`main`, `sub`	overall and sub titles for the plot.
`xlab`	a label for the x axis.
`ylab`	a label for the y axis.
`xlim`	limits for the x axis.
`ylim`	limits for the y axis.
`xpd`	logical. Should bars be allowed to go outside region?
`log`	a character string which contains ‘"x"’ if the x axis is to be logarithmic, ‘"y"’ if the y axis is to be logarithmic and ‘"xy"’ or ‘"yx"’ if both axes are to be logarithmic.
`axes`	logical. If `TRUE`, a vertical (or horizontal, if `horiz` is true) axis is drawn.
`axisnames`	logical. If `TRUE`, and if there are `names.arg` (see above), the other axis is drawn (with `lty = 0`) and labeled.
`cex.axis`	expansion factor for numeric axis labels.
`cex.names`	expansion factor for names.
`inside`	logical. If `TRUE`, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when `space = 0` (which it partly is when `beside = TRUE`).
`plot`	logical. If `FALSE`, nothing is plotted.
`axis.lty`	the graphics parameter `lty` applied to the axis and tick marks of the categorical (default horzontal) axis. Note that by default the axis is suppressed.
`offset`	a vector indicating how much the bars should be shifted relative to the x axis.
`plot.ci`	logical. If `TRUE`, confidence intervals are plotted over the bars. Note that if a stacked bar plot is generated, confidence intervals will not be plotted even if `plot.ci = TRUE`
`ci.l`, `ci.u`	The confidence intervals (ci.l = lower bound, ci.u = upper bound) to be plotted if `plot.ci = TRUE`. Values must have the same dim structure as `height`.
`ci.color`	the color for the confidence interval line segments
`ci.lty`	the line type for the confidence interval line segments
`ci.lwd`	the line width for the confidence interval line segments
`ci.width`	length of lines used for the "t" at the end of confidence interval line segments, as a multple of `width`. Defaults to 0.5.
`plot.grid`	if `TRUE` a lined grid will be plotted behind the bars

`grid.inc`	the number of grid increments to be plotted
`grid.lty`	the line type for the grid
`grid.lwd`	the line width for the grid
`grid.col`	the line color for the grid
`add`	logical, if `TRUE` add barplot to current plot.
`panel.first`	An expression to be evaluated after the plot region coordinates have been set up, but prior to the drawing of the bars and other plot region contents. This can be useful to add additional plot region content behind the bars. This will also work if `add = TRUE`
`panel.last`	An expression to be evaluated after the bars have been drawn, but prior to the addition of confidence intervals, a legend and the axis annotation
`...`	further graphical parameters (`par`) are passed to `plot.window()`, `title()` and `axis`.

Details

This is a generic function, it currently only has a default method. A formula interface may be added eventually.

Value

A numeric vector (or matrix, when `beside = TRUE`), say `mp`, giving the coordinates of *all* the bar midpoints drawn, useful for adding to the graph.

If `beside` is true, use `colMeans(mp)` for the midpoints of each *group* of bars, see example.

Note

Prior to R 1.6.0, `barplot` behaved as if `axis.lty = 1`, unintentionally. 0 (zero) and NA values in `height` will not be plotted if using logarithmic scales. If there are NA values in `height` and `beside = FALSE`, values after the NA will not be plotted in stacked bars.

Author(s)

Original barplot() by R-Core. Enhancements by Marc Schwartz.

See Also

`plot(..., type = "h")`, `dotchart`, `hist`.

Examples

```
tN <- table(Ni <- rpois(100, lambda = 5))
r <- barplot2(tN, col = 'gray')

#- type = "h" plotting *is* `bar'plot
lines(r, tN, type = 'h', col = 'red', lwd = 2)

barplot2(tN, space = 1.5, axisnames = FALSE,
        sub = "barplot2(..., space = 1.5, axisnames = FALSE)")

data(VADeaths, package = "datasets")
barplot2(VADeaths, plot = FALSE)
barplot2(VADeaths, plot = FALSE, beside = TRUE)

mp <- barplot2(VADeaths) # default
tot <- colMeans(VADeaths)
text(mp, tot + 3, format(tot), xpd = TRUE, col = "blue")
barplot2(VADeaths, beside = TRUE,
        col = c("lightblue", "mistyrose", "lightcyan",
                "lavender", "cornsilk"),
        legend = rownames(VADeaths), ylim = c(0, 100))
title(main = "Death Rates in Virginia", font.main = 4)

# Example with confidence intervals and grid
hh <- t(VADeaths)[, 5:1]
mybarcol <- "gray20"
ci.l <- hh * 0.85
ci.u <- hh * 1.15
mp <- barplot2(hh, beside = TRUE,
        col = c("lightblue", "mistyrose",
                "lightcyan", "lavender"),
        legend = colnames(VADeaths), ylim = c(0, 100),
        main = "Death Rates in Virginia", font.main = 4,
        sub = "Faked 95 percent error bars", col.sub = mybarcol,
        cex.names = 1.5, plot.ci = TRUE, ci.l = ci.l, ci.u = ci.u,
        plot.grid = TRUE)
mtext(side = 1, at = colMeans(mp), line = -2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")
box()

# Example with horizontal bars, grid and logarithmic x axis
barplot2(1:10 , log = "x", plot.grid = TRUE, grid.inc = 10,
        xlim = c(0.5, 20), horiz = TRUE, cex.axis = 0.9,
        prcol = "gray95")
box()

# Bar shading example
barplot2(VADeaths, angle = 15 + 10 * 1:5, density = 20, col = "black",
        legend = rownames(VADeaths))
title(main = list("Death Rates in Virginia", font = 4))

# border :
barplot2(VADeaths, border = "dark blue")
```

boxplot2 *Produce a Boxplot Annotated with the Number of Observations*

Description

This funcntion uses `boxplot` to produce a boxplot which is then annotated with the number of observations in each group.

Usage

```
boxplot2(..., top=FALSE, shrink=1, textcolor=NULL)
```

Arguments

`...`	parameters passed to `boxplot`.
`top`	logical indicating whether the number of observations should be added to the top or the bottom of the plotting region. Defaults to `FALSE`.
`shrink`	value to shrink character size (cex) when annotating.
`textcolor`	text color.

Note

This function replaces `boxplot.n`, which has been deprecated avoid potential problems with S3 method dispatching.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

`boxplot`, `text`

Examples

```
data(state)

# n's at bottom
boxplot2( state.area ~ state.region)

# n's at top
boxplot2( state.area ~ state.region, top=TRUE)

# small red text
boxplot2( state.area ~ state.region, shrink=0.8, textcolor="red")
```

bubbleplot *Bubble Plot*

Description

Draw a bubble plot, a scatterplot with varying symbol sizes and colors, or add points to existing plots. A variety of input formats are supported, including vectors, matrices, data frames, formulas, etc.

Usage

```
bubbleplot(x, ...)

## Default S3 method:
bubbleplot(x, y, z, std=TRUE, pow=0.5, add=FALSE,
           rev=FALSE, type="p", ylim=NULL, xlab=NULL, ylab=NULL,
           pch=c(16,1), cex.points=1, col="black", bg=par("bg"), ...)

## S3 method for class 'formula'
bubbleplot(formula, data, subset, na.action=NULL, ...)
```

Arguments

`x`	a vector of values for the horizontal axis. Can also be a 2-dimensional matrix or table (x values in column names and y values in row names), or a data frame containing `x`, `y`, and `z` in that order. If the data frame contains column names `x`, `y`, and `z` then they will be used for plotting.
`...`	passed to `plot` and `points`.
`y`	a vector of values for the vertical axis.
`z`	a vector of values determining the bubble sizes.
`std`	whether to standardize the `z` values.
`pow`	a power coefficient for the bubble sizes.
`add`	whether to add bubbles to an existing plot.
`rev`	whether to reverse the `y` axis.
`type`	passed to `points`.
`ylim`	passed to `plot`.
`xlab`, `ylab`	passed to `plot`.
`pch`	passed to `points`.
`cex.points`	scales all bubble sizes.
`col`, `bg`	passed to `points`.
`formula`	has the form `z ~ x + y`, where `z` determines the bubble sizes and `x` and `y` determine bubble locations.
`data`	where formula terms are stored, e.g. data frame or list.
`subset`	a logical vector specifying which data to plot.
`na.action`	how `NA` values are handled.

Details

The std standardization sets z = abs(z) / mean(abs(z)).

The pow = 0.5 (square root) is a good default, where a z value of 2 has twice the area of 1. See example #2 below for an exception, where the z value is tree circumference and therefore proportional to the tree diameter.

The pch, col, and bg arguments can be be vectors of length 2, where positive z values are drawn with pch[1], col[1], bg[1] and negative z values are drawn with pch[2], col[2], and bg[2].

Author(s)

Arni Magnusson.

See Also

points is the underlying function used to draw the bubbles.

symbols can also draw bubbles, but does not handle negative z values or have convenience features such as pow and rev.

balloonplot provides an alternative interface and visual style based on tables instead of scatterplots.

Examples

```
catch.t <- xtabs(Catch~Year+Age, catch.d)              # example table
catch.m <- as.matrix(as.data.frame(unclass(catch.t)))  # example matrix

# 1  Formula
bubbleplot(Catch~Age+Year, data=catch.d)
# Use rev=TRUE to get same layout as crosstab matrix:
print(catch.m)
bubbleplot(Catch~Age+Year, data=catch.d, rev=TRUE, las=1)

# 2  Data frame
bubbleplot(catch.d)
bubbleplot(Orange)
# Visualize tree transverse section at breast height
bubbleplot(Orange, pow=1, cex=2, pch=21,
           col="darkred", bg="peru", lwd=1.5)

# 3  Matrix or table
bubbleplot(catch.m)
bubbleplot(catch.t)

# 4  Positive and negative values
bubbleplot(catch.r)
bubbleplot(Resid~Age+Year, catch.r, subset=Age %in% 4:9,
           rev=TRUE, xlim=c(3.5,9.5), cex=1.3)
# Residuals from orange tree model
library(nlme)
fm <- nlme(circumference~phi1/(1+exp(-(age-phi2)/phi3)),
           fixed=phi1+phi2+phi3~1, random=phi1~1|Tree,
           data=Orange, start=c(phi1=200,phi2=800,phi3=400))
bubbleplot(residuals(fm)~Tree+age, Orange)
bubbleplot(residuals(fm)~Tree+age, Orange, cex=2.5, pch=16,
           col=c("dodgerblue","orange"))

# 5  Richter magnitude, amplitude, and energy release
bubbleplot(mag~long+lat, quakes, pch=1)
bubbleplot(10^mag~long+lat, quakes, cex=1.2, col=gray(0, 0.3))
bubbleplot(sqrt(1000)^mag~long+lat, quakes, cex=1.2, col=gray(0, 0.3))
bubbleplot(sqrt(1000)^mag~long+lat, quakes, cex=1.2, col="#FF00004D")
```

`catch.d` *Catch at Age and Residuals*

Description

Catch-at-age observed data and model residuals from Icelandic saithe assessment.

Usage

```
catch.d
catch.r
```

Format

Data frame containing three columns:

`Year`	year
`Age`	age
and	
`Catch`	catch (thousands of individuals)
or	
`Resid`	standardized residual

Details

The data are from Tables 8.2 and 8.6 in the ICES (2015) fish stock assessment of Icelandic saithe.

Source

ICES (2015) Report of the North-Western Working Group (NWWG). *ICES CM 2015/ACOM:07*, pp. 240–246.

See Also

`bubbleplot` is an effective way to visualize these data.

Examples

```
catch.t <- xtabs(Catch~Year+Age, catch.d)
catch.m <- as.matrix(as.data.frame(unclass(catch.t)))

# 1  Formula
bubbleplot(Catch~Age+Year, data=catch.d)
# Use rev=TRUE to get same layout as crosstab matrix:
print(catch.m)
bubbleplot(Catch~Age+Year, data=catch.d, rev=TRUE, las=1)

# 2  Data frame
bubbleplot(catch.d)

# 3  Matrix or table
bubbleplot(catch.m)
bubbleplot(catch.t)

# 4  Positive and negative values
bubbleplot(catch.r)
bubbleplot(Resid~Age+Year, catch.r, subset=Age %in% 4:9,
           rev=TRUE, xlim=c(3.5,9.5), cex=1.3)
```

---

ci2d *Create 2-dimensional empirical confidence regions*

Description

Create 2-dimensional empirical confidence regions from provided data.

Usage

```
ci2d(x, y = NULL,
     nbins=51, method=c("bkde2D","hist2d"),
     bandwidth, factor=1.0,
     ci.levels=c(0.50,0.75,0.90,0.95,0.975),
     show=c("filled.contour","contour","image","none"),
     col=topo.colors(length(breaks)-1),
     show.points=FALSE,
     pch=par("pch"),
     points.col="red",
     xlab, ylab,
     ...)
## S3 method for class 'ci2d'
print(x, ...)
```

Arguments

x	either a vector containing the x coordinates or a matrix with 2 columns.

`y`	a vector contianing the y coordinates, not required if ‘x’ is matrix
`nbins`	number of bins in each dimension. May be a scalar or a 2 element vector. Defaults to 51.
`method`	One of "bkde2D" (for KernSmooth::bdke2d) or "hist2d" (for gplots::hist2d) specifyting the name of the method to create the 2-d density summarizing the data. Defaults to "bkde2D".
`bandwidth`	Bandwidth to use for `KernSmooth::bkde2D`. See below for default value.
`factor`	Numeric scaling factor for bandwidth. Useful for exploring effect of changing the bandwidth. Defaults to 1.0.
`ci.levels`	Confidence level(s) to use for plotting data. Defaults to `c(0.5, 0.75, 0.9, 0.95, 0.975)`
`show`	Plot type to be displaed. One of "filled.contour", "contour", "image", or "none". Defaults to "filled.contour".
`show.points`	Boolean indicating whether original data values should be plotted. Defaults to `TRUE`.
`pch`	Point type for plots. See `points` for details.
`points.col`	Point color for plotting original data. Defaiults to "red".
`col`	Colors to use for plots.
`xlab, ylab`	Axis labels
`...`	Additional arguments passed to `KernSmooth::bkde2D` or `gplots::hist2d`.

Details

This function utilizes either `KernSmooth::bkde2D` or `gplots::hist2d` to estmate a 2-dimensional density of the data passed as an argument. This density is then used to create and (optionally) display confidence regions.

When `bandwidth` is ommited and `method="bkde2d"`, `KernSmooth::dpik` is appled in `x` and `y` dimensions to select the bandwidth.

Value

A `ci2d` object consisting of a list containing (at least) the following elements:

`nobs`	number of original data points
`x`	x position of each density estimate bin
`y`	y position of each density estimate bin
`density`	Matrix containing the probability density of each bin (count in bin/total count)
`cumDensity`	Matrix where each element contains the cumulative probability density of all elements with the same density (used to create the confidence region plots)
`contours`	List of contours of each confidence region.
`call`	Call used to create this object

Note

Confidence intervals generated by ci2d are *approximate*, and are subject to biases and/or artifacts induced by the binning or kernel smoothing method, bin locations, bin sizes, and kernel bandwidth.

The `conf2d` function in the **r2d2** package may create a more accurate confidence region, and reports the actual proportion of points inside the region.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`bkde2D`, `conf2d`, `dpik`, `hist2d`

Examples

```
####
## Basic usage
####
data(geyser, package="MASS")

x <- geyser$duration
y <- geyser$waiting

# 2-d confidence intervals based on binned kernel density estimate
ci2d(x,y)                   # filled contour plot
ci2d(x,y, show.points=TRUE) # show original data


# image plot
ci2d(x,y, show="image")
ci2d(x,y, show="image", show.points=TRUE)

# contour plot
ci2d(x,y, show="contour", col="black")
ci2d(x,y, show="contour", col="black", show.points=TRUE)

####
## Control Axis scales
####
x <- rnorm(2000, sd=4)
y <- rnorm(2000, sd=1)

# 2-d confidence intervals based on binned kernel density estimate
ci2d(x,y)

# 2-d confidence intervals based on 2d histogram
ci2d(x,y, method="hist2d", nbins=25)

# Require same scale for each axis, this looks oval
ci2d(x,y, range.x=list(c(-20,20), c(-20,20)))
```

```
ci2d(x,y, method="hist2d", same.scale=TRUE, nbins=25) # hist2d

####
## Control smoothing and binning
####
x <- rnorm(2000, sd=4)
y <- rnorm(2000, mean=x, sd=2)

# Default 2-d confidence intervals based on binned kernel density estimate
ci2d(x,y)

# change the smoother bandwidth
ci2d(x,y,
     bandwidth=c(sd(x)/8, sd(y)/8)
    )

# change the smoother number of bins
ci2d(x,y, nbins=10)
ci2d(x,y)
ci2d(x,y, nbins=100)

# Default 2-d confidence intervals based on 2d histogram
ci2d(x,y, method="hist2d", show.points=TRUE)

# change the number of histogram bins
ci2d(x,y, nbin=10, method="hist2d", show.points=TRUE )
ci2d(x,y, nbin=25, method="hist2d", show.points=TRUE )

####
## Perform plotting manually
####
data(geyser, package="MASS")

# let ci2d handle plotting contours...
ci2d(geyser$duration, geyser$waiting, show="contour", col="black")

# call contour() directly, show the 90 percent CI, and the mean point
est <- ci2d(geyser$duration, geyser$waiting, show="none")
contour(est$x, est$y, est$cumDensity,
        xlab="duration", ylab="waiting",
        levels=0.90, lwd=4, lty=2)
points(mean(geyser$duration), mean(geyser$waiting),
      col="red", pch="X")


####
## Extract confidence region values
###
data(geyser, package="MASS")

## Empirical 90 percent confidence limits
quantile( geyser$duration, c(0.05, 0.95) )
quantile( geyser$waiting, c(0.05, 0.95) )
```

```
## Bivariate 90 percent confidence region
est <- ci2d(geyser$duration, geyser$waiting, show="none")
names(est$contours) ## show available contours

ci.90 <- est$contours[names(est$contours)=="0.9"]  # get region(s)
ci.90 <- rbind(ci.90[[1]],NA, ci.90[[2]], NA, ci.90[[3]]) # join them

print(ci.90)                  # show full contour
range(ci.90$x, na.rm=TRUE)    # range for duration
range(ci.90$y, na.rm=TRUE)    # range for waiting

####
## Visually compare confidence regions
####
data(geyser, package="MASS")

## Bivariate smoothed 90 percent confidence region
est <- ci2d(geyser$duration, geyser$waiting, show="none")
names(est$contours) ## show available contours

ci.90 <- est$contours[names(est$contours)=="0.9"]  # get region(s)
ci.90 <- rbind(ci.90[[1]],NA, ci.90[[2]], NA, ci.90[[3]]) # join them

plot( waiting ~ duration, data=geyser,
      main="Comparison of 90 percent confidence regions" )
polygon( ci.90, col="green", border="green", density=10)

## Univariate Normal-Theory 90 percent confidence region
mean.x <- mean(geyser$duration)
mean.y <- mean(geyser$waiting)
sd.x <- sd(geyser$duration)
sd.y <- sd(geyser$waiting)

t.value <- qt(c(0.05,0.95), df=length(geyser$duration), lower=TRUE)
ci.x <- mean.x +  t.value* sd.x
ci.y <- mean.y +  t.value* sd.y

plotCI(mean.x, mean.y,
       li=ci.x[1],
       ui=ci.x[2],
       barcol="blue", col="blue",
       err="x",
       pch="X",
       add=TRUE )

plotCI(mean.x, mean.y,
       li=ci.y[1],
       ui=ci.y[2],
       barcol="blue", col="blue",
       err="y",
       pch=NA,
       add=TRUE )
```

```
#   rect(ci.x[1], ci.y[1], ci.x[2], ci.y[2], border="blue",
#        density=5,
#        angle=45,
#        col="blue" )


   ## Empirical univariate 90 percent confidence region
   box <- cbind( x=quantile( geyser$duration, c(0.05, 0.95 )),
                 y=quantile( geyser$waiting, c(0.05, 0.95 )) )

   rect(box[1,1], box[1,2], box[2,1], box[2,2], border="red",
        density=5,
        angle=-45,
        col="red" )

   ## now a nice legend
   legend( "topright", legend=c("       Region type",
                                "Univariate Normal Theory",
                                "Univarite Empirical",
                                "Smoothed Bivariate"),
           lwd=c(NA,1,1,1),
           col=c("black","blue","red","green"),
           lty=c(NA,1,1,1)
         )

   ####
   ## Test with a large number of points
   ####
   ## Not run:
   x <- rnorm(60000, sd=1)
   y <- c( rnorm(40000, mean=x, sd=1),
           rnorm(20000, mean=x+4, sd=1) )

   hist2d(x,y)
   ci <- ci2d(x,y)
   ci

## End(Not run)
```

col2hex *Convert color names to hex RGB strings*

Description

Convert color names to hex RGB strings

Usage

```
col2hex(cname)
```

Arguments

`cname`	Color name(s)

Value

Character vector giving the hex color code translation of the provided color names.

Author(s)

Gregory R. Warnes

See Also

`col2rgb`, `colors`, `rgb`

Examples

```
col2hex(c("red","yellow","lightgrey"))
```

`colorpanel` *Generate a smoothly varying set of colors*

Description

`colorpanel` generate a set of colors that varies smoothly. `redgreen`, `greenred`, `bluered`, and `redblue` generate red-black-green, green-black-red, red-white-blue, and blue-white-red colorbars, respectively. colors

Usage

```
colorpanel(n, low, mid, high)
redgreen(n)
greenred(n)
bluered(n)
redblue(n)
```

Arguments

`n`	Desired number of color elements in the panel.
`low`, `mid`, `high`	Colors to use for the Lowest, middle, and highest values. `mid` may be ommited.

Details

The values for `low`, `mid`, `high` can be given as color names (`"red"`), plot color index (`2`), and HTML-style RGB, (`"#FF0000"`).

If `mid` is supplied, then the returned color panel will consist of `n - floor(n/2)` HTML-style RGB elements which vary smoothly between `low` and `mid`, then between `mid` and `high`. Note that if `n` is even, the color `mid` will occur twice at the center of the sequence.

If `mid` is omitted, the color panel will vary smoothly beween `low` and `high`.

Value

Vector of HTML-style RGB colors.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

colors

Examples

```
showpanel <- function(col)
{
  image(z=matrix(1:100, ncol=1), col=col, xaxt="n", yaxt="n" )
}

par(mfrow=c(3,3))

# two colors only:
showpanel(colorpanel(8,low="red",high="green"))

# three colors
showpanel(colorpanel(8,"red","black","green"))
# note the duplicatation of black at the center, using an odd
# number of elements resolves this:
showpanel(colorpanel(9,"red","black","green"))

showpanel(greenred(64))
showpanel(redgreen(64))
showpanel(bluered(64))
showpanel(redblue(64))
```

---

gplots-defunct *Defunct functions*

Description

These functions are defunct and have been removed from the gplots package.

Usage

```
boxplot.n(..., top=FALSE, shrink=1, textcolor=NULL)
plot.lm2(
         x,
         which = 1:5,
         caption = c("Residuals vs Fitted", "Normal Q-Q plot",
                  "Scale-Location plot", "Cook's distance plot"),
                panel = panel.smooth,
                sub.caption = deparse(x$call),
                main = "",
                ask,
                ...,
                id.n = 3,
                labels.id = names(residuals(x)),
                cex.id = 0.75,
                band=TRUE,
                rug=TRUE,
                width=1/10,
                max.n=5000
                )
      smartlegend(x = c("left", "center", "right"),
                 y = c("top", "center", "bottom"),
                 ...,
                 inset = 0.05)
```

Arguments

`ask`, `band`, `caption`, `cex.id`, `id.n`, `inset`, `labels.id`, `main`, `max.n`, `panel`, `rug`, `shrink`, `sub.caption`, `textcolor`, `top`, `which`, `width`, `x`, `y`, `...`
see man page for the corresponding replacement function

Details

These functions are no longer available. Please refer to the manual page for the replacement function:

- `boxplot.n` has been replaced by `boxplot2`
- `plot.lm2` has been replaced by `lmplot2`
- `smartlegend` is no longer needed because relative positioning has been implemented in `legend`.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`boxplot2`, `lmplot2`, `legend`, `Defunct`

gplots-deprecated — *Deprecated functions*

Description

These functions have been deprecated and will be removed in future releases of gplots.

Usage

```
## No deprecated functions at this time ##
```

Details

These functions have been deprecated. Please refer to the manual page for the replacement function:

- (No deprecated functions at this time)

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

Deprecated

heatmap.2 — *Enhanced Heat Map*

Description

A heat map is a false color image (basically `image(t(x))`) with a dendrogram added to the left side and/or to the top. Typically, reordering of the rows and columns according to some set of values (row or column means) within the restrictions imposed by the dendrogram is carried out.

This heatmap provides a number of extensions to the standard R `heatmap` function.

Usage

```
heatmap.2 (x,

           # dendrogram control
           Rowv = TRUE,
           Colv=if(symm)"Rowv" else TRUE,
           distfun = dist,
           hclustfun = hclust,
           dendrogram = c("both","row","column","none"),
           reorderfun = function(d, w) reorder(d, w),
           symm = FALSE,

           # data scaling
           scale = c("none","row", "column"),
           na.rm=TRUE,

           # image plot
           revC = identical(Colv, "Rowv"),
           add.expr,

           # mapping data to colors
           breaks,
           symbreaks=any(x < 0, na.rm=TRUE) || scale!="none",

           # colors
           col="heat.colors",

           # block sepration
           colsep,
           rowsep,
           sepcolor="white",
           sepwidth=c(0.05,0.05),

           # cell labeling
           cellnote,
           notecex=1.0,
           notecol="cyan",
           na.color=par("bg"),

           # level trace
           trace=c("column","row","both","none"),
           tracecol="cyan",
           hline=median(breaks),
           vline=median(breaks),
           linecol=tracecol,

           # Row/Column Labeling
           margins = c(5, 5),
           ColSideColors,
           RowSideColors,
           cexRow = 0.2 + 1/log10(nr),
           cexCol = 0.2 + 1/log10(nc),
           labRow = NULL,
           labCol = NULL,
           srtRow = NULL,
           srtCol = NULL,
           adjRow = c(0,NA),
           adjCol = c(NA,0),
           offsetRow = 0.5,
           offsetCol = 0.5,
           colRow = NULL,
           colCol = NULL,

           # color key + density info
           key = TRUE,
           keysize = 1.5,
           density.info=c("histogram","density","none"),
           denscol=tracecol,
           symkey = any(x < 0, na.rm=TRUE) || symbreaks,
           densadj = 0.25,
           key.title = NULL,
           key.xlab = NULL,
           key.ylab = NULL,
           key.xtickfun = NULL,
           key.ytickfun = NULL,
           key.par=list(),

           # plot labels
           main = NULL,
           xlab = NULL,
           ylab = NULL,

           # plot layout
           lmat = NULL,
           lhei = NULL,
           lwid = NULL,

           # extras
           extrafun=NULL,
           ...
           )
```

Arguments

`x`	numeric matrix of the values to be plotted.
`Rowv`	determines if and how the *row* dendrogram should be reordered. By default, it is TRUE, which implies dendrogram is computed and reordered based on row means. If NULL or FALSE, then no dendrogram is computed and no reordering is done. If a `dendrogram`, then it is used "as-is", ie without any reordering. If a vector of integers, then dendrogram is computed and reordered based on the order of the vector.
`Colv`	determines if and how the *column* dendrogram should be reordered. Has the options as the `Rowv` argument above and *additionally* when `x` is a square matrix, `Colv="Rowv"` means that columns should be treated identically to the rows.

`distfun`	function used to compute the distance (dissimilarity) between both rows and columns. Defaults to `dist`.
`hclustfun`	function used to compute the hierarchical clustering when `Rowv` or `Colv` are not dendrograms. Defaults to `hclust`.
`dendrogram`	character string indicating whether to draw 'none', 'row', 'column' or 'both' dendrograms. Defaults to 'both'. However, if Rowv (or Colv) is FALSE or NULL and dendrogram is 'both', then a warning is issued and Rowv (or Colv) arguments are honoured.
`reorderfun`	`function(d, w)` of dendrogram and weights for reordering the row and column dendrograms. The default uses `stats{reorder.dendrogram}`

.

`symm`	logical indicating if `x` should be treated **symm**etrically; can only be true when `x` is a square matrix.
`scale`	character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. The default is `"none"`.
`na.rm`	logical indicating whether `NA`'s should be removed.
`revC`	logical indicating if the column order should be `rev`ersed for plotting, such that e.g., for the symmetric case, the symmetry axis is as usual.
`add.expr`	expression that will be evaluated after the call to `image`. Can be used to add components to the plot.
`breaks`	(optional) Either a numeric vector indicating the splitting points for binning `x` into colors, or a integer number of break points to be used, in which case the break points will be spaced equally between `min(x)` and `max(x)`.
`symbreaks`	Boolean indicating whether breaks should be made symmetric about 0. Defaults to `TRUE` if the data includes negative values, and to `FALSE` otherwise.
`col`	colors used for the image. Defaults to heat colors (`heat.colors`).
`colsep`, `rowsep`, `sepcolor`	(optional) vector of integers indicating which columns or rows should be separated from the preceding columns or rows by a narrow space of color `sepcolor`.
`sepwidth`	(optional) Vector of length 2 giving the width (colsep) or height (rowsep) the separator box drawn by colsep and rowsep as a function of the width (colsep) or height (rowsep) of a cell. Defaults to `c(0.05, 0.05)`
`cellnote`	(optional) matrix of character strings which will be placed within each color cell, e.g. p-value symbols.
`notecex`	(optional) numeric scaling factor for `cellnote` items.
`notecol`	(optional) character string specifying the color for `cellnote` text. Defaults to "cyan".
`na.color`	Color to use for missing value (`NA`). Defaults to the plot background color.
`trace`	character string indicating whether a solid "trace" line should be drawn across 'row's or down 'column's, 'both' or 'none'. The distance of the line from the center of each color-cell is proportional to the size of the measurement. Defaults to 'column'.

`tracecol`	character string giving the color for "trace" line. Defaults to "cyan".
`hline`, `vline`, `linecol`	Vector of values within cells where a horizontal or vertical dotted line should be drawn. The color of the line is controlled by `linecol`. Horizontal lines are only plotted if `trace` is 'row' or 'both'. Vertical lines are only drawn if `trace` 'column' or 'both'. `hline` and `vline` default to the median of the breaks, `linecol` defaults to the value of `tracecol`.
`margins`	numeric vector of length 2 containing the margins (see `par(mar= *)`) for column and row names, respectively.
`ColSideColors`	(optional) character vector of length `ncol(x)` containing the color names for a horizontal side bar that may be used to annotate the columns of `x`.
`RowSideColors`	(optional) character vector of length `nrow(x)` containing the color names for a vertical side bar that may be used to annotate the rows of `x`.
`cexRow`, `cexCol`	positive numbers, used as `cex.axis` in for the row or column axis labeling. The defaults currently only use number of rows or columns, respectively.
`labRow`, `labCol`	character vectors with row and column labels to use; these default to `rownames(x)` or `colnames(x)`, respectively.
`srtRow`, `srtCol`	angle of row/column labels, in degrees from horizontal
`adjRow`, `adjCol`	2-element vector giving the (left-right, top-bottom) justification of row/column labels (relative to the text orientation).
`offsetRow`, `offsetCol`	Number of character-width spaces to place between row/column labels and the edge of the plotting region.
`colRow`, `colCol`	color of row/column labels, either a scalar to set the color of all labels the same, or a vector providing the colors of each label item
`key`	logical indicating whether a color-key should be shown.
`keysize`	numeric value indicating the size of the key
`density.info`	character string indicating whether to superimpose a 'histogram', a 'density' plot, or no plot ('none') on the color-key.
`denscol`	character string giving the color for the density display specified by `density.info`, defaults to the same value as `tracecol`.
`symkey`	Boolean indicating whether the color key should be made symmetric about 0. Defaults to `TRUE` if the data includes negative values, and to `FALSE` otherwise.
`densadj`	Numeric scaling value for tuning the kernel width when a density plot is drawn on the color key. (See the `adjust` parameter for the `density` function for details.) Defaults to 0.25.
`key.title`	main title of the color key. If set to NA no title will be plotted.
`key.xlab`	x axis label of the color key. If set to NA no label will be plotted.
`key.ylab`	y axis label of the color key. If set to NA no label will be plotted.
`key.xtickfun`	function computing tick location and labels for the xaxis of the color key. Returns a named list containing parameters that can be passed to `axis`. See examples.

`key.ytickfun`	function computing tick location and labels for the y axis of the color key. Returns a named list containing parameters that can be passed to `axis`. See examples.
`key.par`	graphical parameters for the color key. Named list that can be passed to `par`.
`main, xlab, ylab`	main, x- and y-axis titles; defaults to none.
`lmat, lhei, lwid`	visual layout: position matrix, column height, column width. See below for details
`extrafun`	A function to be called after all other work. See examples.
`...`	additional arguments passed on to `image`

Details

If either `Rowv` or `Colv` are dendrograms they are honored (and not reordered). Otherwise, dendrograms are computed as `dd <- as.dendrogram(hclustfun(distfun(X)))` where `X` is either `x` or `t(x)`.

If either is a vector (of "weights") then the appropriate dendrogram is reordered according to the supplied values subject to the constraints imposed by the dendrogram, by `reorder(dd, Rowv)`, in the row case. If either is missing, as by default, then the ordering of the corresponding dendrogram is by the mean value of the rows/columns, i.e., in the case of rows, `Rowv <- rowMeans(x, na.rm=na.rm)`. If either is `NULL`, *no reordering* will be done for the corresponding side.

If `scale="row"` (or `scale="col"`) the rows (columns) are scaled to have mean zero and standard deviation one. There is some empirical evidence from genomic plotting that this is useful.

The default colors range from red to white (`heat.colors`) and are not pretty. Consider using enhancements such as the **RColorBrewer** package, `https://cran.r-project.org/package=RColorBrewer` to select better colors.

By default four components will be displayed in the plot. At the top left is the color key, top right is the column dendrogram, bottom left is the row dendrogram, bottom right is the image plot. When RowSideColor or ColSideColor are provided, an additional row or column is inserted in the appropriate location. This layout can be overriden by specifiying appropriate values for `lmat`, `lwid`, and `lhei`. `lmat` controls the relative postition of each element, while `lwid` controls the column width, and `lhei` controls the row height. See the help page for `layout` for details on how to use these arguments.

Value

Invisibly, a list with components

`rowInd`	row index permutation vector as returned by `order.dendrogram`.
`colInd`	column index permutation vector.
`call`	the matched call
`rowMeans, rowSDs`	mean and standard deviation of each row: only present if `scale="row"`
`colMeans, colSDs`	mean and standard deviation of each column: only present if `scale="column"`

`carpet`	reordered and scaled 'x' values used generate the main 'carpet'
`rowDendrogram`	row dendrogram, if present
`colDendrogram`	column dendrogram, if present
`breaks`	values used for color break points
`col`	colors used
`vline`	center-line value used for column trace, present only if `trace="both"` or `trace="column"`
`hline`	center-line value used for row trace, present only if `trace="both"` or `trace="row"`
`colorTable`	A three-column data frame providing the lower and upper bound and color for each bin
`layout`	A named list containing the values used for `lmat`, `lhei`, and `lwid`.

Note

The original rows and columns are reordered to match the dendrograms `Rowv` and `Colv` (if present).

`heatmap.2()` uses `layout` to arragent the plot elements. Consequentially, it can **not** be used in a multi column/row layout using `layout(...)`, `par(mfrow=...)` or `(mfcol=...)`.

Author(s)

Andy Liaw, original; R. Gentleman, M. Maechler, W. Huber, G. Warnes, revisions.

See Also

`image`, `hclust`

Examples

```
data(mtcars)
x  <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start=0, end=.3)
cc <- rainbow(ncol(x), start=0, end=.3)

##
## demonstrate the effect of row and column dendrogram options
##
heatmap.2(x)                    ## default - dendrogram plotted and reordering done.
heatmap.2(x, dendrogram="none") ##  no dendrogram plotted, but reordering done.
heatmap.2(x, dendrogram="row")  ## row dendrogram plotted and row reordering done.
heatmap.2(x, dendrogram="col")  ## col dendrogram plotted and col reordering done.

heatmap.2(x, keysize=2)         ## default - dendrogram plotted and reordering done.

heatmap.2(x, Rowv=FALSE, dendrogram="both") ## generates a warning!
heatmap.2(x, Rowv=NULL, dendrogram="both")  ## generates a warning!
heatmap.2(x, Colv=FALSE, dendrogram="both") ## generates a warning!

## Reorder dendrogram by branch means rather than sums
heatmap.2(x, reorderfun=function(d, w) reorder(d, w, agglo.FUN = mean) )
```

```
## plot a sub-cluster using the same color coding as for the full heatmap
full <- heatmap.2(x)
heatmap.2(x, Colv=full$colDendrogram[[2]], breaks=full$breaks)  # column subset
heatmap.2(x, Rowv=full$rowDendrogram[[1]], breaks=full$breaks)  # row subset
heatmap.2(x, Colv=full$colDendrogram[[2]],
             Rowv=full$rowDendrogram[[1]], breaks=full$breaks)  # both

## Show effect of row and column label rotation
heatmap.2(x, srtCol=NULL)
heatmap.2(x, srtCol=0,   adjCol = c(0.5,1) )
heatmap.2(x, srtCol=45,  adjCol = c(1,1)   )
heatmap.2(x, srtCol=135, adjCol = c(1,0)   )
heatmap.2(x, srtCol=180, adjCol = c(0.5,0) )
heatmap.2(x, srtCol=225, adjCol = c(0,0)   ) ## not very useful
heatmap.2(x, srtCol=270, adjCol = c(0,0.5) )
heatmap.2(x, srtCol=315, adjCol = c(0,1)   )
heatmap.2(x, srtCol=360, adjCol = c(0.5,1) )

heatmap.2(x, srtRow=45, adjRow=c(0, 1) )
heatmap.2(x, srtRow=45, adjRow=c(0, 1), srtCol=45, adjCol=c(1,1) )
heatmap.2(x, srtRow=45, adjRow=c(0, 1), srtCol=270, adjCol=c(0,0.5) )


## Show effect of offsetRow/offsetCol (only works when srtRow/srtCol is
## not also present)
heatmap.2(x, offsetRow=0, offsetCol=0)
heatmap.2(x, offsetRow=1, offsetCol=1)
heatmap.2(x, offsetRow=2, offsetCol=2)
heatmap.2(x, offsetRow=-1, offsetCol=-1)

heatmap.2(x, srtRow=0, srtCol=90, offsetRow=0, offsetCol=0)
heatmap.2(x, srtRow=0, srtCol=90, offsetRow=1, offsetCol=1)
heatmap.2(x, srtRow=0, srtCol=90, offsetRow=2, offsetCol=2)
heatmap.2(x, srtRow=0, srtCol=90, offsetRow=-1, offsetCol=-1)


## Show how to use 'extrafun' to replace the 'key' with a scatterplot
lmat <- rbind( c(5,3,4), c(2,1,4) )
lhei <- c(1.5, 4)
lwid <- c(1.5, 4, 0.75)

myplot <- function() {
            oldpar <- par("mar")
            par(mar=c(5.1, 4.1, 0.5, 0.5))
            plot(mpg ~ hp, data=x)
          }

heatmap.2(x, lmat=lmat, lhei=lhei, lwid=lwid, key=FALSE, extrafun=myplot)

## show how to customize the color key
heatmap.2(x,
          key.title=NA, # no title
          key.xlab=NA,  # no xlab
          key.par=list(mgp=c(1.5, 0.5, 0),
                       mar=c(2.5, 2.5, 1, 0)),
          key.xtickfun=function() {
                breaks <- parent.frame()$breaks
                return(list(
                     at=parent.frame()$scale01(c(breaks[1],
                                                 breaks[length(breaks)])),
                     labels=c(as.character(breaks[1]),
                              as.character(breaks[length(breaks)]))
                     ))
          })

heatmap.2(x,
         breaks=256,
         key.title=NA,
         key.xlab=NA,
         key.par=list(mgp=c(1.5, 0.5, 0),
                      mar=c(1, 2.5, 1, 0)),
         key.xtickfun=function() {
              cex <- par("cex")*par("cex.axis")
              side <- 1
              line <- 0
              col <- par("col.axis")
              font <- par("font.axis")
              mtext("low", side=side, at=0, adj=0,
                    line=line, cex=cex, col=col, font=font)
              mtext("high", side=side, at=1, adj=1,
                    line=line, cex=cex, col=col, font=font)
              return(list(labels=FALSE, tick=FALSE))
         })


##
## Show effect of z-score scaling within columns, blue-red color scale
##
hv <- heatmap.2(x, col=bluered, scale="column", tracecol="#303030")

###
## Look at the return values
###
names(hv)

## Show the mapping of z-score values to color bins
hv$colorTable

## Extract the range associated with white
hv$colorTable[hv$colorTable[,"color"]=="#FFFFFF",]

## Determine the original data values that map to white
whiteBin <- unlist(hv$colorTable[hv$colorTable[,"color"]=="#FFFFFF",1:2])
rbind(whiteBin[1] * hv$colSDs + hv$colMeans,
      whiteBin[2] * hv$colSDs + hv$colMeans )
```

```
##
## A more decorative heatmap, with z-score scaling along columns
##
hv <- heatmap.2(x, col=cm.colors(255), scale="column",
      RowSideColors=rc, ColSideColors=cc, margin=c(5, 10),
      xlab="specification variables", ylab= "Car Models",
      main="heatmap(<Mtcars data>, ..., scale=\"column\")",
        tracecol="green", density="density")
## Note that the breakpoints are now symmetric about 0

## Color the labels to match RowSideColors and ColSideColors
hv <- heatmap.2(x, col=cm.colors(255), scale="column",
        RowSideColors=rc, ColSideColors=cc, margin=c(5, 10),
      xlab="specification variables", ylab= "Car Models",
      main="heatmap(<Mtcars data>, ..., scale=\"column\")",
        tracecol="green", density="density", colRow=rc, colCol=cc,
        srtCol=45, adjCol=c(0.5,1))




data(attitude)
round(Ca <- cor(attitude), 2)
symnum(Ca) # simple graphic

# with reorder
heatmap.2(Ca,   symm=TRUE, margin=c(6, 6), trace="none" )

# without reorder
heatmap.2(Ca, Rowv=FALSE, symm=TRUE, margin=c(6, 6), trace="none" )

## Place the color key below the image plot
heatmap.2(x, lmat=rbind( c(0, 3), c(2,1), c(0,4) ), lhei=c(1.5, 4, 2 ) )

## Place the color key to the top right of the image plot
heatmap.2(x, lmat=rbind( c(0, 3, 4), c(2,1,0 ) ), lwid=c(1.5, 4, 2 ) )

## For variable clustering, rather use distance based on cor():
data(USJudgeRatings)
symnum( cU <- cor(USJudgeRatings) )

hU <- heatmap.2(cU, Rowv=FALSE, symm=TRUE, col=topo.colors(16),
             distfun=function(c) as.dist(1 - c), trace="none")

## The Correlation matrix with same reordering:
hM <- format(round(cU, 2))
hM

# now with the correlation matrix on the plot itself

heatmap.2(cU, Rowv=FALSE, symm=TRUE, col=rev(heat.colors(16)),
            distfun=function(c) as.dist(1 - c), trace="none",
            cellnote=hM)
```

```
## genechip data examples
## Not run:
library(affy)
data(SpikeIn)
pms <- SpikeIn@pm

# just the data, scaled across rows
heatmap.2(pms, col=rev(heat.colors(16)), main="SpikeIn@pm",
             xlab="Relative Concentration", ylab="Probeset",
             scale="row")

# fold change vs "12.50" sample
data <- pms / pms[, "12.50"]
data <- ifelse(data>1, data, -1/data)
heatmap.2(data, breaks=16, col=redgreen, tracecol="blue",
              main="SpikeIn@pm Fold Changes\nrelative to 12.50 sample",
              xlab="Relative Concentration", ylab="Probeset")

## End(Not run)
```

hist2d *Compute and Plot a 2-Dimensional Histogram*

Description

Compute and plot a 2-dimensional histogram.

Usage

```
hist2d(x,y=NULL, nbins=200, same.scale=FALSE, na.rm=TRUE, show=TRUE,
       col=c("black", heat.colors(12)), FUN=base::length, xlab, ylab,
       ... )
## S3 method for class 'hist2d'
print(x, ...)
```

Arguments

`x`	either a vector containing the x coordinates or a matrix with 2 columns.
`y`	a vector contianing the y coordinates, not required if ‘x’ is matrix
`nbins`	number of bins in each dimension. May be a scalar or a 2 element vector. Defaults to 200.
`same.scale`	use the same range for x and y. Defaults to FALSE.
`na.rm`	Indicates whether missing values should be removed. Defaults to TRUE.
`show`	Indicates whether the histogram be displayed using `image` once it has been computed. Defaults to TRUE.

`col`	Colors for the histogram. Defaults to "black" for bins containing no elements, a set of 16 heat colors for other bins.
`FUN`	Function used to summarize bin contents. Defaults to `base::length`. Use, e.g., `mean` to calculate means for each bin instead of counts.
`xlab`, `ylab`	(Optional) x and y axis labels
`...`	Parameters passed to the image function.

Details

This fucntion creates a 2-dimensional histogram by cutting the x and y dimensions into `nbins` sections. A 2-dimensional matrix is then constucted which holds the counts of the number of observed (x,y) pairs that fall into each bin. If `show=TRUE`, this matrix is then then passed to `image` for display.

Value

A list containing 5 elements:

`counts`	Matrix containing the number of points falling into each bin
`x.breaks`, `y.breaks`	Lower and upper limits of each bin
`x`, `y`	midpoints of each bin

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

`image`, `persp`, `hist`, `freq2d`

Examples

```
## example data, bivariate normal, no correlation
x <- rnorm(2000, sd=4)
y <- rnorm(2000, sd=1)

## separate scales for each axis, this looks circular
hist2d(x,y)

## same scale for each axis, this looks oval
hist2d(x,y, same.scale=TRUE)

## use different ## bins in each dimension
hist2d(x,y, same.scale=TRUE, nbins=c(100,200) )

## use the hist2d function to create an h2d object
h2d <- hist2d(x,y,show=FALSE, same.scale=TRUE, nbins=c(20,30))

## show object summary
```

```
h2d

## object contents
str(h2d)

## perspective plot
persp( h2d$x, h2d$y, h2d$counts,
       ticktype="detailed", theta=30, phi=30,
       expand=0.5, shade=0.5, col="cyan", ltheta=-30)

## for contour (line) plot ...
contour( h2d$x, h2d$y, h2d$counts, nlevels=4 )

## for a filled contour plot ...
filled.contour( h2d$x, h2d$y, h2d$counts, nlevels=4,
                col=gray((4:0)/4) )
```

`lmplot2` *Plots to assess the goodness of fit for the linear model objects*

Description

Plots to assess the goodness of fit for the linear model objects

Usage

```
lmplot2(
         x,
         which = 1:5,
         caption = c("Residuals vs Fitted", "Normal Q-Q plot",
           "Scale-Location plot", "Cook's distance plot"),
         panel = panel.smooth,
         sub.caption = deparse(x$call),
         main = "",
         ask = interactive() && nb.fig < length(which)
         && .Device != "postscript",
         ...,
         id.n = 3,
         labels.id = names(residuals(x)),
         cex.id = 0.75,
         band=TRUE,
         rug=TRUE,
         width=1/10,
         max.n=5000
         )
```

Arguments

`x`	lm object
`which`	Numerical values between 1 and 5, indicating which plots to be shown. The codes are: **1** Fitted vs residuals **2** Normal Q-Q **3** Scale-Location **4** Cook's distance **5** Residuals vs. predictor
`caption`	Caption for each type of plot
`panel`	function to draw on the existing plot
`sub.caption`	SubCaption for the plots
`main`	Main title of the plot
`ask`	whether interactive graphics
`...`	parameters passed to `lmplot2`.
`id.n`	integer value, less than or equal to residuals of lm object
`labels.id`	Names of the residuals of the lm object
`cex.id`	Parameter to control the height of text stringsx
`band`	logical vector indicating whether bandplot should also be plotted
`rug`	logical vector indicating whether rug should be added to the existing plot
`width`	Fraction of the data to use for plot smooths
`max.n`	Maximum number of points to display in plots

Note

This function replaces `plot.lm2`, which has been deprecated to avoid potential problems with S3 method dispatching.

Author(s)

Gregory R. Warnes <`greg@warnes.net`> and Nitin Jain <`nitin.jain@pfizer.com`>

See Also

`plot.lm`

Examples

```
ctl <- rnorm(100, 4)
trt <- rnorm(100, 4.5)
group <- gl(2,100,200, labels=c("Ctl","Trt"))
weight <- c(ctl, trt)
wt.err <- rnorm(length(weight), mean=weight, sd=1/2)
x <- lm(weight ~ group + wt.err)
```

```
lmplot2(x)

lmplot2(x, which=1,   width=1/3)
lmplot2(x, which=1:3, width=1/3)
```

---

lowess *Scatter Plot Smoothing*

Description

The lowess function performs the computations for the *LOWESS* smoother (see the reference below). lowess returns a an object containing components x and y which give the coordinates of the smooth. The smooth can then be added to a plot of the original points with the function lines.

Alternatively, plot can be called directly on the object returned from lowess and the 'lowess' method for plot will generate a scatterplot of the original data with a lowess line superimposed.

Finally, the plotLowess function both calculates the lowess smooth and plots the original data with a lowess smooth.

Usage

```
lowess(x, ...)

## Default S3 method:
lowess(x, y=NULL, f=2/3, iter=3L, delta=0.01 *
       diff(range(x)), ...)

## S3 method for class 'formula'
lowess(formula,data=parent.frame(), ..., subset, f=2/3,
       iter=3L, delta=.01*diff(range(mf[-response])))

## S3 method for class 'lowess'
plot(x, y, ..., col.lowess="red", lty.lowess=2)

plotLowess(formula, data=parent.frame(), ..., subset=parent.frame(),
           col.lowess="red", lty.lowess=2  )
```

Arguments

formula	formula providing a single dependent variable (y) and an single independent variable (x) to use as coordinates in the scatter plot.
data	a data.frame (or list) from which the variables in 'formula' should be taken.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
x, y	vectors giving the coordinates of the points in the scatter plot. Alternatively a single plotting structure can be specified.

`f`	the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness.
`iter`	the number of robustifying iterations which should be performed. Using smaller values of `iter` will make `lowess` run faster.
`delta`	values of `x` which lie within `delta` of each other replaced by a single value in the output from `lowess`.
`...`	parameters for methods.
`col.lowess`, `lty.lowess`	color and line type for plotted line

References

Cleveland, W. S. (1979) Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.* **74**, 829–836.

Cleveland, W. S. (1981) LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, **35**, 54.

See Also

`loess` (in package `modreg`), a newer formula based version of `lowess` (with different defaults!).

Examples

```
data(cars)

#
# x,y method
#
plot(cars$speed, cars$dist, main="lowess(cars)")
lines(lowess(cars$speed, cars$dist), col=2)
lines(lowess(cars$speed, cars$dist, f=.2), col=3)
legend(5, 120, c(paste("f=", c("2/3", ".2"))), lty=1, col=2:3)

#
# formula method: plot, then calculate the lowess smoother,
#                 then add smooth to the plot
#
plot(dist ~ speed, data=cars, main="lowess(cars)")
lines(lowess(dist ~ speed, data=cars), col=2, lty=2)
lines(lowess(dist ~ speed, data=cars, f=.2), col=3) # smaller bandwith
legend(5, 120, c(paste("f=", c("2/3", ".2"))), lty=1, col=2:3)

#
# formula method: calculate lowess() smoother, then call plot()
#                  on the lowess object
#
lw <- lowess(dist ~ speed, data=cars)
plot(lw, main="lowess(cars)"  )

#
```

```
# formula method: calculate and plot in a single command
#
plotLowess(dist ~ speed, data=cars, main="lowess(cars)")
```

---

ooplot.default *Create an OpenOffice style plot*

Description

An extension of barplot2. Creates bar- and line-plots mimicking the style of OpenOffice plots. This utility can plot the values next to each point or bar as well as confidence intervals.

Usage

```
ooplot(data, ...)
## Default S3 method:
ooplot(data, width=1, space=NULL, names.arg=NULL,
                              legend.text=NULL, horiz=FALSE,
                              density=NULL, angle=45, kmg="fpnumkMGTP",
                              kmglim=TRUE,
                              type=c("xyplot", "linear", "barplot", "stackbar"),
                              col=heat.colors(NC), prcol=NULL,
                              border=par("fg"), main=NULL, sub=NULL,
                              xlab=NULL, ylab=NULL, xlim=NULL, ylim=NULL,
                              xpd=TRUE, log="", axes=TRUE,
                              axisnames=TRUE, prval=TRUE, lm=FALSE,
                              cex.axis=par("cex.axis"),
                              cex.names=par("cex.axis"),
                              cex.values=par("cex"),inside=TRUE,
                              plot=TRUE, axis.lty=0, plot.ci=FALSE,
                              ci.l=NULL, ci.u=NULL, ci.color="black",
                              ci.lty="solid", ci.lwd=1, plot.grid=FALSE,
                              grid.inc=NULL, grid.lty="dotted",
                              grid.lwd=1, grid.col="black", add=FALSE,
                              by.row=FALSE, ...)
```

Arguments

`data`	a matrix of values describing the values that make up the plot. The first column of `data` is taken as the axis against which all the other values are plotted. The first column of `data` may not be sparse.
`width`	optional vector of barwidths. Re-cycled to the number of bars drawn. A single value will have no visible effect.

`space`	the amount of space left before each bar. May be given as a single number or one number per bar. If `type` is `stackbar`, `space` may be specified by two numbers, where the first is the space between bars in the same group, and the second the space between groups. Defaults to `c(0,1)` if `type` is a `stackbar`, and to 0.2 otherwise.
`names.arg`	a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the row names of `data`.
`legend.text`	a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included; if `legend.text` is true, the row names of `data` will be used as labels if they are non-null.
`horiz`	a logical value. If `FALSE`, the bars are drawn vertically with the first bar to the left. If `TRUE`, the bars are drawn horizontally with the first at the bottom.
`density`	a vector giving the the density of shading lines, in lines per inch, for the bars or bar components. The default value of `NULL` means that no shading lines are drawn. Non-positive values of `density` also inhibit the drawing of shading lines.
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise), for the bars or bar components.
`kmg`	the set of SI units to convert, defaults to "fpnumkMGTP". See below for details.
`kmglim`	logical. If `FALSE` the conversion to SI units is not performed. Default is `TRUE`.
`type`	a string indicating the preferred format of the plot, choices are: xyplot : plot where y is plotted against the x-value. linear : plot where y values are plotted against equidistant x-values. barplot : plot where y values are represented as bars against equidistant x-values. stackplot : plot where y values are stacked for identical x-values and bars are equidistant.
`col`	a vector of colors for the bars or bar components.
`prcol`	the color to be used for the plot region.
`border`	the color to be used for the border of the bars.
`main`, `sub`	overall and sub titles for the plot.
`xlab`	a label for the x axis.
`ylab`	a label for the y axis.
`xlim`	limits for the x axis.
`ylim`	limits for the y axis.
`xpd`	logical. Should bars be allowed to go outside region?
`log`	a character string which contains ‘"x"’ if the x axis is to be logarithmic, ‘"y"’ if the y axis is to be logarithmic and ‘"xy"’ or ‘"yx"’ if both axes are to be logarithmic.
`axes`	logical. If `TRUE`, a vertical (or horizontal, if `horiz` is true) axis is drawn.
`axisnames`	logical. If `TRUE`, and if there are `names.arg` (see above), the other axis is drawn (with `lty=0`) and labeled.
`prval`	logical. If `TRUE`, then values are plotted above all points and bars.
`lm`	logical. If `TRUE`, the linear fit is plotted.

`cex.axis`, `cex.names`, `cex.values`	character scaling factor for numeric axis labels, names, and displayed values, respectively.
`inside`	logical. If `TRUE`, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when `space = 0` (which it partly is when `beside = TRUE`).
`plot`	logical. If `FALSE`, nothing is plotted.
`axis.lty`	the graphics parameter `lty` applied to the axis and tick marks of the categorical (default horzontal) axis. Note that by default the axis is suppressed.
`plot.ci`	logical. If `TRUE`, confidence intervals are plotted over the bars. Note that if a stacked bar plot is generated, confidence intervals will not be plotted even if `plot.ci = TRUE`
`ci.l`, `ci.u`	The confidence intervals (ci.l = lower bound, ci.u = upper bound) to be plotted if `plot.ci = TRUE`. Values must have the same dim structure as `height`.
`ci.color`	the color for the confidence interval line segments
`ci.lty`	the line type for the confidence interval line segments
`ci.lwd`	the line width for the confidence interval line segments
`plot.grid`	if `TRUE` a lined grid will be plotted behind the bars
`grid.inc`	the number of grid increments to be plotted
`grid.lty`	the line type for the grid
`grid.lwd`	the line width for the grid
`grid.col`	the line color for the grid
`add`	logical, if `TRUE` add barplot to current plot.
`by.row`	Logical value. If `TRUE` the data matrix is organized with variables along rows rather than down colums.
`...`	further graphical parameters (`par`) are passed to `plot.window()`, `title()` and `axis`.

Details

Plot units are automatically scaled to SI units based on the maximum value present, according to the set of units specified by characters in the `kmg` parameter. These letters are interpreted as

P	peta = 1E15
T	tera = 1E12
G	giga = 1E09
M	mega = 1E06
k	kilo = 1E03
m	milli= 1E-03
u	micro= 1E-06
n	nano = 1E-09
p	pico = 1E-12
f	femto= 1E-15

with the default being "fpnumkMGTP" (all of these units). For example, if the largest value plotted is 1243000, it would be presented as 1.234M.

Value

A numeric vector (or matrix, when `beside = TRUE`), say `mp`, giving the coordinates of *all* the bar midpoints drawn, useful for adding to the graph.

If `beside` is true, use `colMeans(mp)` for the midpoints of each *group* of bars, see example.

Author(s)

Lodewijk Bonebakker <bonebakker@comcast.net> with modifications by Gregory R. Warnes <greg@warnes.net>. Based on barplot2().

See Also

`plot`, `boxplot`

Examples

```
data(VADeaths, package = "datasets")

VADeaths <- cbind( Age=c(50,55,60,65,70), VADeaths)

mp <- ooplot(VADeaths) # default
mp <- ooplot(VADeaths, type="xyplot")  # same as default
mp <- ooplot(VADeaths, type="linear")  # linear scale
mp <- ooplot(VADeaths, type="linear", log="y") # log scale on y axis
mp <- ooplot(VADeaths, type="barplot") # barplot
mp <- ooplot(VADeaths, type="stackbar") # stacked


tot <- colMeans(VADeaths[,-1])
ooplot(VADeaths,
        col = c("lightblue", "mistyrose", "lightcyan", "lavender"),
        legend = colnames(VADeaths)[-1], ylim = c(0, 100),
        type="barplot", cex.values=0.75)
title(main = "Death Rates in Virginia", font.main = 4)


##
## Capability demo
##
## examples for the ooplot routine
##
## create some test data
test1 <- data.frame(x=c(0,1,2,3,4), lin=c(0,1,2,3,4))
test2 <- data.frame(x=c(0,1,2,3,4), par=c(0,1,4,9,16))
test3 <- data.frame(x=c(-2,-1,0,1,2),y2=c(4,1,0,1,4))
## single line test example
test1f <- test1
## two column example
test2f <- merge(test1,test2,by.x="x",all=TRUE,sort=TRUE)
## three column example
test3f <- merge(test2f,test3,by.x="x",all=TRUE,sort=TRUE)
```

```
## subset, single row, example
test5r <- test3f[5,]

##
## xyplot, linear, barplot, stackbar
dev.off()
mat <- matrix(c(1:16),4,4,byrow=TRUE)
layout(mat)

ooplot(test1f,type="barplot",col=c("red"))
title(main="barplot")
ooplot(test2f,type="barplot",col=c("red","blue"))
ooplot(test3f,type="barplot",col=c("red","blue","green"))
ooplot(test5r,type="barplot",col=c("red","blue","green"))

ooplot(test1f,type="xyplot",col=c("red"))
title(main="xyplot")
ooplot(test2f,type="xyplot",col=c("red","blue"))
ooplot(test3f,type="xyplot",col=c("red","blue","green"))
ooplot(test5r,type="xyplot",col=c("red","blue","green"))

ooplot(test1f,type="linear",col=c("red"))
title(main="linear")
ooplot(test2f,type="linear",col=c("red","blue"))
ooplot(test3f,type="linear",col=c("red","blue","green"))
ooplot(test5r,type="linear",col=c("red","blue","green"))

ooplot(test1f,type="stackbar",col=c("red"))
title(main="stackbar")
ooplot(test2f,type="stackbar",col=c("red","blue"))
ooplot(test3f,type="stackbar",col=c("red","blue","green"))
ooplot(test5r,type="stackbar",col=c("red","blue","green"))

# restore default layout (1 plot/page)
layout(1)
```

overplot — *Plot multiple variables on the same region, with appropriate axes*

Description

overplot graphs a set of variables defined on the same x-range but which have varying y-ranges on the same plotting area. For each set of y-values it uses a different color and line-type and and draws a correspondingly colored and line-typed axis. panel.overplot is used by overplot to draw the individual graphs.

Usage

```
overplot(formula, data = parent.frame(), same.scale = FALSE, xlab, ylab,
         xlim, ylim, min.y, max.y, log = "", panel = "panel.overplot",
         subset, plot = TRUE, groups, main, f = 2/3, ...)
```

Arguments

`formula`	Formula describing the x and y variables. It should be of the form x ~ y|z. The conditioning variable (z) should be a factor.
`same.scale`	Logical value indicating whether the plot region should have the same range for all plots. Defaults to `FALSE`.
`xlab`, `ylab`, `xlim`, `ylim`, `main`	Standard plotting parameters. See `plot` for details
`min.y`, `max.y`	Scalar or vector values used to specify the y plotting limits for individual plots. If a single scalar value is provided, it will be used for all plots. These parameters can be used specify one end of the individual plot ranges, while allowing the other end to vary with the data. EG, to force 0 to always be within the plot region.
`log`	character string ”, 'x', 'y', or 'xy', indicating which axes should be plotted on a log scale. Defaults to ” (neither).
`panel`	a plotting function to be called to draw the individual plots. Defaults to `overplot.panel`, which plots the points and a `lowess` smooth.
`plot`	Logical value indicating whether to draw the plot.
`groups`	(optional) character vector giving the names of levels of the conditioning variable to plot. Defaults to all levels of the conditioning variable.
`f`	Smoothing parameter for `lowess`
`data`, `subset`, `...`	parameters passed to `model.frame` to obtain the data to be plotted from the formula.

Details

This function essentially performs

tmp <- split(data, z)

for(i in levels(z))

plot(x ~ y, data=tmp[[z]])

except that all of the plots are shown on the same plotting region and varying scales for each value of z are handled nicely.

Value

A copy of the data split by the conditioning variable.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

`interaction.plot`, `coplot` for alternative visualizations of 3-way data.

Examples

```
# Example teratogenicity rtPCR data
data(rtPCR)

# same scale
overplot( RQ ~ Conc..ug.ml. | Test.Substance,
         data=rtPCR,
         subset=Detector=="ProbeType 1" & Conc..ug.ml. > 0,
         same.scale=TRUE,
         log="xy",
         f=3/4,
         main="Detector=ProbeType 1",
         xlab="Concentration (ug/ml)",
         ylab="Relative Gene Quantification"
         )

# different scales, but force lower limit to 0.01
overplot( RQ ~ Conc..ug.ml. | Test.Substance,
         data=rtPCR,
         subset=Detector=="ProbeType 8" & Conc..ug.ml. > 0,
         log="xy",
         f=3/4,
         main="Detector=ProbeType 8",
         xlab="Concentration (ug/ml)",
         ylab="Relative Gene Quantification",
         min.y=0.01
         )
```

`plotCI` *Plot Error Bars and Confidence Intervals*

Description

Given a set of x and y values and interval width or upper and lower bounds, plot the points with error bars. This can be a useful tool for visualizing confidence intervals.

Usage

```
plotCI(x, y = NULL, uiw, liw = uiw, ui, li, err='y', ylim=NULL,
       xlim=NULL, type="p",  col=par("col"), barcol=col,
       pt.bg = par("bg"),  sfrac = 0.01, gap=1, lwd=par("lwd"),
       lty=par("lty"), labels=FALSE, add=FALSE, xlab, ylab,  minbar,
       maxbar, ... )
```

Arguments

`x, y`	coordinates for the center of error bars. `y` defaults to `1:n`.
`uiw`	width of the upper or right error bar. Set to `NULL` or `NA` to omit upper bars.

`liw`	width of the lower or left error bar. Defaults to same value as `uiw`. Set to `NULL` or `NA` to omit lower bars.
`ui`	upper end of error bars. Defaults to `y + uiw` or `x + uiw` depeding on `err`. Set to `NULL` or `NA` to omit upper bars.
`li`	lower end of error bars. Defaults to `y - liw` or `x - liw` depedning on `err`. Set to `NULL` or `NA` to omit lower bars.
`err`	direction for error bars. Set to "y" for vertical bars. Set to "x" for horizontal bars. Defaults to "y".
`col`	color of plotting character used center marker of error bars. Default is "black".
`xlim`, `ylim`	range of x/y values to include in the plotting area.
`type`	point/line type; passed to `points`
`barcol`	color of the error bars. Defaults to the same value as `col`
`pt.bg`	background color of points (use `pch=21, pt.bg=par("bg")` to get open points superimposed on error bars).
`sfrac`	width of "crossbar" at the end of error bar as a fraction of the x plotting region. Defaults to 0.01.
`gap`	space left between the center of the error bar and the lines marking the error bar in units of the height (width) of the letter "O". Defaults to 1.0
`lwd`	width of bar lines.
`lty`	line type of bar lines.
`labels`	either a logical value indicating whether the circles representing the x values should be replaced with text giving the actual values or a vector containing labels to use instead. Defaults to `FALSE`.
`add`	logical indicating whether error bars should be added to the current plot. If `FALSE` (the default), a new plot will be created and symbols/labels for the x values will be plotted before drawing error bars.
`minbar`	minumum allowed value for bar ends. If specified, values smaller than `minbar` will be replaced with `minbar`.
`maxbar`	maximum allowed value for bar ends. If specified, values larger than `maxbar` will be replaced with `maxbar`.
`...`	optional plotting parameters
`xlab`	label for x axis.
`ylab`	label for y axis.

Author(s)

Original version by Bill Venables <`wvenable@attunga.stats.adelaide.edu.au`> posted to r-help on Sep. 20, 1997. Enhanced version posted to r-help by Ben Bolker <`ben@zoo.ufl.edu`> on Apr. 16, 2001. This version was modified and extended by Gregory R. Warnes <`greg@warnes.net`>. Additional changes suggested by Martin Maechler <`maechler@stat.math.ethz.ch`> integrated on July 29, 2004.

See Also

`plotmeans` provides an enhanced wrapper to `plotCI`.

Examples

```
# plot means and
data(state)
tmp   <- split(state.area, state.region)
means <- sapply(tmp, mean)
stdev <- sqrt(sapply(tmp, var))
n     <- sapply(tmp,length)
ciw   <- qt(0.975, n) * stdev / sqrt(n)

# plain
plotCI(x=means, uiw=ciw)

# prettier
plotCI(x=means, uiw=ciw, col="black", barcol="blue", lwd=1)

# give mean values
plotCI(x=means, uiw=ciw, col="black", barcol="blue",
       labels=round(means,-3), xaxt="n", xlim=c(0,5) )
axis(side=1, at=1:4, labels=names(tmp), cex=0.7)

# better yet, just use plotmeans ... #
plotmeans( state.area ~ state.region )
```

---

plotmeans *Plot Group Means and Confidence Intervals*

Description

Plot group means and confidence intervals.

Usage

```
plotmeans(formula, data=NULL, subset, na.action,
          bars=TRUE, p=0.95, minsd=0, minbar, maxbar,
          xlab=names(mf)[2], ylab=names(mf)[1], mean.labels=FALSE,
          ci.label=FALSE, n.label=TRUE, text.n.label="n=",
          digits=getOption("digits"), col="black", barwidth=1,
          barcol="blue", connect=TRUE, ccol=
          col, legends=names(means), xaxt, use.t=TRUE,
          lwd=par("lwd"), ...)
```

Arguments

`formula`	symbolic expression specifying the outcome (continuous) and grouping variable (factor). See lm() for details.
`data`	optional data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain 'NA's. See lm() for details.
`bars`	a logical value indicating whether confidence interval bars should be plotted. Defaults to TRUE.
`p`	confidence level for error bars. Defaults to 0.95.
`minsd`	minumum permitted value for the standard deviation within each factor level. Any standard deviation estimates smaller than `minsd` will be replaced with `minsd`. Defaults to 0.
`minbar`	minumum allowed value for bar ends. If specified, values smaller than `minbar` will be replaced with `minbar`.
`maxbar`	maximum allowed value for bar ends. If specified, values larger than `maxbar` will be replaced with `maxbar`.
`xlab`	x-axis label.
`ylab`	y-axis label.
`mean.labels`	either a logical value indicating whether the circles representing the group means should be replaced with text giving the actual mean values or a vector containing labels to use instead. Defaults to FALSE.
`ci.label`	a logical value indicating whether text giving the actual interval end values should be placed at the end of each confidence interval bar. Defaults to FALSE.
`n.label`	a logical value indicating whether text giving the number of observations in each group should should be added to the plot.
`text.n.label`	Prefix text for labeling observation counts. Defaults to "n=".
`digits`	number of significant digits to use when displaying mean or confidince limit values.
`col`	color of cicles marking group means. Default is "black".
`barwidth`	linewidth of interval bars and end marks. Default is 1.
`barcol`	color of interval bars and end marks. Default is "blue".
`connect`	either a logical value indicating whether the means of each group should be connected by a line, or a list of vectors giving the index of bars that should be connected by a line. Defaults to TRUE.
`ccol`	color of lines used to connect means. Defaults to the same color as "col".
`legends`	vector containing strings used to label groups along the x axis. Defaults to group names.
`xaxt`	A character which specifies the axis type. Specifying '"n"' causes an axis to be set up, but not plotted.

`use.t`	a logical value indicating whether the t distribution should be used to compute confidence intervals. If `TRUE`, the default, a t distribution will the correct number of degrees of freedom for each group be used. If `FALSE`, the a normal distribution will be used.
`lwd`	Width of connecting lines
`...`	optional plotting parameters.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`plotCI`, `boxplot`

Examples

```
# library(gplots)
# show comparison with boxplot
data(state)
plotmeans(state.area ~ state.region)

# show some color and mean labels
plotmeans(state.area ~ state.region,
           mean.labels=TRUE, digits=-3,
           col="red", connect=FALSE)

# show how to specify which means should be connected
plotmeans(state.area ~ state.region, connect=list(1:2, 3:4),
          ccol="red", pch=7 )

# more complicated example showing how to show an interaction
data(esoph)
par(las=2,                          # use perpendicular axis labels
    mar=c(10.1,4.1,4.1,2.1),        # create enough space for long x labels
    mgp=c(8,1,0)                    # move x axis legend down to avoid overlap
    )
plotmeans(ncases/ncontrols ~ interaction(agegp , alcgp, sep ="   "),
          connect=list(1:6,7:12,13:18,19:24),
          barwidth=2,
          col="dark green",
          data=esoph,
          xlab="Age Group and Alcohol Consumption",
          ylab="# Cases / # Controls",
          ylim = c(-.9,1.4),
          main=c("Fraction of Cases for by Age and Alcohol Consumption",
                 "Ile-et-Vilaine Esophageal Cancer Study")
          )
abline(v=c(6.5, 12.5, 18.5), lty=2)
```

qqnorm.aov *Makes a half or full normal plot for the effects from an aov model*

Description

Makes a half or full normal plot for the effects from a model inheriting from class aov. One can interactively label the points in the plot.

Usage

```
## S3 method for class 'aov'
qqnorm(y, full=FALSE, label=FALSE, omit=NULL,
          xlab=paste(if (full) "" else "Half", " Normal plot"),
          ylab="Effects", ...)
```

Arguments

y	A model object inheriting from aov
full	Full or half normal plot (half is default)
label	If TRUE, function allows interactive labelling of points in plot, using the mouse
omit	Numeric or character vector of effects to omit, the intercept is always omitted
xlab	Horizontal axix label
ylab	Vertical axis label
...	Further arguments to be given to the plot function

Details

Produces a (half) normal plot of the effects from an AOV model. The idea behind the plot is that most effects will be small or null, and this effects can be used as a basis for estimation of the experimental variance. This small effects will show up in the plot as a straight line, other effects can be judged against this as a background. Heavily used by Box, Hunter & Hunter, which attributes the idea to Daniel.

Value

If `label=TRUE`, the vector of points identified, else nothing of interest.

Author(s)

Kjetil Halvorsen <kjetil@entelnet.bo>

References

Box, Hunter and Hunter: Statistics for Experimenters. An Introduction to Design, Data Analysis and Model Building. Wiley.
Daniel, C (1976): Applications of Statistics to Industrial Experimentation. Wiley.
Daniel, C (1959): Use of half-normal plot in interpreting factorial two-level experiments. *Technometrics.***1**, 149.

Examples

```
library(MASS)
data(npk)
npk.aov <- aov(yield ~ block + N*P*K, npk)
qqnorm(npk.aov)

## interactive labeling of points.  Click mouse on points to show label.
if (dev.interactive()) qqnorm(npk.aov, omit=2:6, label=TRUE)
```

`reorder.factor` *Reorder the Levels of a Factor*

Description

Reorder the levels of a factor

Usage

```
## S3 method for class 'factor'
reorder(x, X, FUN, ..., order=is.ordered(x), new.order, sort=mixedsort)
```

Arguments

`x`	factor
`X`	auxillary data vector
`FUN`	function to be applied to subsets of `X` determined by `x`, to determine factor order
`...`	optional parameters to `FUN`
`order`	logical value indicating whether the returned object should be an `ordered` factor
`new.order`	a vector of indexes or a vector of label names giving the order of the new factor levels
`sort`	function to use to sort the factor level names, used only when `new.order` is missing

Details

This function changes the order of the levels of a factor. It can do so via three different mechanisms, depending on whether, `X` *and* `FUN`, `new.order` or `sort` are provided.

If `X` *and* `Fun` are provided: The data in `X` is grouped by the levels of `x` and `FUN` is applied. The groups are then sorted by this value, and the resulting order is used for the new factor level names.

If `new.order` is a numeric vector, the new factor level names are constructed by reordering the factor levels according to the numeric values. If `new.order` is a chraccter vector, `new.order` gives the list of new factor level names. In either case levels omitted from `new.order` will become missing (`NA`) values.

If `sort` is provided (as it is by default): The new factor level names are generated by calling the function specified by `sort` to the existing factor level *names*. With `sort=mixedsort` (the default) the factor levels are sorted so that combined numeric and character strings are sorted in according to character rules on the character sections (including ignoring case), and the numeric rules for the numeric sections. See `mixedsort` for details.

Value

A new factor with reordered levels

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`factor` and `reorder`

Examples

```
# Create a 4 level example factor
trt <- factor( sample( c("PLACEBO", "300 MG", "600 MG", "1200 MG"),
               100, replace=TRUE ) )
summary(trt)
# Note that the levels are not in a meaningful order.

# Change the order to something useful..
# - default "mixedsort" ordering
trt2 <- reorder(trt)
summary(trt2)
# - using indexes:
trt3 <- reorder(trt, new.order=c(4, 2, 3, 1))
summary(trt3)
# - using label names:
trt4 <- reorder(trt, new.order=c("PLACEBO", "300 MG", "600 MG", "1200 MG"))
summary(trt4)
# - using frequency
trt5 <- reorder(trt, X=rnorm(100), FUN=mean)
summary(trt5)

# Drop out the '300 MG' level
trt6 <- reorder(trt, new.order=c("PLACEBO", "600 MG", "1200 MG"))
summary(trt6)
```

`residplot` *Undocumented functions*

Description

These functions are undocumented. Some are internal and not intended for direct use. Others simply haven't been documented yet.

Usage

```
residplot(model, formula, ...)
```

Arguments

`model`	Undocumented
`formula`	Undocumented
`...`	arguments to be passed to `fun`

Details

These functions are undocumented. Some are internal and not intended for direct use. Others simply haven't been documented yet.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

`rich.colors` *Rich Color Palettes*

Description

Create a vector of `n` colors that are perceptually equidistant and in an order that is easy to interpret.

Usage

```
rich.colors(n, palette="temperature", alpha=1.0, rgb=FALSE, plot=FALSE)
```

Arguments

`n`	number of colors to generate.
`palette`	palette to use: `"temperature"` contains blue-green-yellow-red, and `"blues"` contains black-blue-white.
`alpha`	alpha transparency, from 0 (fully transparent) to 1 (opaque).
`rgb`	if `TRUE` then a matrix of RGBA values is included as an attribute.
`plot`	whether to plot a descriptive color diagram.

Value

A character vector of color codes.

Author(s)

Arni Magnusson.

See Also

rgb, rainbow, heat.colors.

Examples

```
m <- abs(matrix(1:120+rnorm(120), nrow=15, ncol=8))
opar <- par(bg="gray", mfrow=c(1,2))
matplot(m, type="l", lty=1, lwd=3, col=rich.colors(8))
matplot(m, type="l", lty=1, lwd=3, col=rich.colors(8,"blues"))
par(opar)

barplot(rep(1,100), col=rich.colors(100), space=0, border=0, axes=FALSE)
barplot(rep(1,20), col=rich.colors(40)[11:30]) # choose subset

plot(m, rev(m), ylim=c(120,0), pch=16, cex=2,
     col=rich.colors(200,"blues",alpha=0.6)[1:120]) # semitransparent

rich.colors(100, plot=TRUE)  # describe rgb recipe

par(mfrow=c(2,2))
barplot(m, col=heat.colors(15), main="\nheat.colors")
barplot(m, col=1:15, main="\ndefault palette")
barplot(m, col=rich.colors(15), main="\nrich.colors")
barplot(m, col=rainbow(15), main="\nrainbow")
par(opar)
```

---

rtPCR *Teratogenesis rtPCR data*

Description

rtPCR data for experiments investigating a variety of markers for characterizing teratogenicity.

Usage

```
data(rtPCR)
```

Format

A data frame with 1672 observations on the following 21 variables.

PlateID a factor with levels A0027002 through A0054019

Test.Substance a factor with levels Compound A through Compound H

Teratogenicity.in.vivo a factor with levels Non Strong Weak / Moderate

Sample a factor with levels Sample 1 - Sample 152

Rep.. a factor with levels Rep 1 - Rep 21

Label a factor with levels Ctrl, Neg. Ctrl P1 - P9, No Vehicle Ctrl, and Pos. Ctrl

Conc..ug.ml. a numeric vector

Detector a factor with levels ProbeType 1 - ProbeType 17

Avg.delta.Ct a numeric vector

delta.Ct.SD a numeric vector

delta.delta.Ct a numeric vector

RQ a numeric vector

X..RQ a numeric vector

X100..Custom.. a numeric vector

X100...Custom.. a numeric vector

Custom.. a numeric vector

Custom...1 a numeric vector

RQ.Min a numeric vector

RQ.Max a numeric vector

Threshold a numeric vector

Details

TBA

Source

Anonymized data.

Examples

```
data(rtPCR)

# same scale
overplot( RQ ~ Conc..ug.ml. | Test.Substance,
         data=rtPCR,
         subset=Detector=="ProbeType 7" & Conc..ug.ml. > 0,
         same.scale=TRUE,
         log="xy",
         f=3/4,
         main="Detector=ProbeType 7",
         xlab="Concentration (ug/ml)",
         ylab="Relative Gene Quantification"
         )

# different scales, but force lower limit to 0.01
overplot( RQ ~ Conc..ug.ml. | Test.Substance,
         data=rtPCR,
         subset=Detector=="ProbeType 7" & Conc..ug.ml. > 0,
         log="xy",
         f=3/4,
         main="Detector=ProbeType 7",
         xlab="Concentration (ug/ml)",
         ylab="Relative Gene Quantification",
         min.y=0.01
         )
```

`sinkplot` *Send textual R output to a graphics device*

Description

Divert R's standard text output to a graphics device.

Usage

```
sinkplot(operation = c("start", "plot", "cancel"), ...)
```

Arguments

`operation`	See below
`...`	Plot arguments. (Ignored unless `operation`="plot").

Details

This function allows the printed output of R commands to be captured and displayed on a graphics device.

The capture process is started by calling `sinkplot("start")`. Now R commands can be executed and all printed output (except errors) will be captured. When the desired text has been captured `sinkplot("plot")` can be called to actually display the output. `sinkplot("cancel")` can be used to abort the output capture without plotting.

The current implementation does not allow `sinkplot` to be nested.

Value

Invisibly returns a character vector containing one element for each line of the captured output.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

References

Functionality requested by Kevin Wright <`kwright@eskimo.com`> in the R-devel newlist posting `https://stat.ethz.ch/pipermail/r-devel/2004-January/028483.html`.

See Also

`capture.output`, `textplot`

Examples

```
set.seed(12456)
x <- factor(sample( LETTERS[1:5], 50, replace=TRUE))
y <- rnorm(50, mean=as.numeric(x), sd=1)

## construct a figure showing a box plot of the data, followed by an
## analysis of variance table for the data
layout(cbind(1:2), heights=c(2,1))

boxplot(y~x, col="darkgreen")

sinkplot()
anova(lm(y~x))
sinkplot("plot",col="darkgreen")
```

---

space *Space points in an x-y plot so they don't overlap.*

Description

Space points in an x-y plot so they don't overlap.

Usage

```
space(x, y, s=1/50, na.rm=TRUE, direction="x")
```

Arguments

`x`	numeric vector of x coordonates.
`y`	numeric vector of x coordonates.
`s`	either a single numeric value or 2 element vector specifying the minimum distance between points in the x and y dimensions as a fraction of the x and y range. Defaults to 1/50.
`na.rm`	logical indicating whether pairs where one or both elements are missing should be removed. Defaults to TRUE.
`direction`	"x" or "y", indicating which direction points should be moved to accomplish spacine.

Details

In an x-y plot where at least one variable has discrete levels several points may be plotted at or very near the same coordonates. This makes it difficult to guage the number of points in a specific region. A common method of resolving this problem is to 'jitter' the points by adding random noise.

This function takes a different approach to the same problem.

When there are two or more points with the same (x,y) value (or within x+-s[1] and x+-s[2]), it spaces these out in the x direction so that the points are separated by at least distance s.

Another method for dealing with overploting is available in the `sunflowerplot` function.

Value

list with two components

x	(modified) x location for each input point
y	y location of each input point

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

jitter, sunflowerplot

Examples

```
x <- rep(1:5, 10)
y <- round(rnorm(length(x),x))

prepar <- par("mfrow")
par(mfrow=c(1,3))

# standard x-y plot: noverlapping points are hidden
plot(x,y)
title("Standard Plot")

# 'spaced' plot: overlapping points are spread out and visible
plot(space(x,y))
title("Plot with 'space'")


# 'spaced' plot: overlapping points are spread out along y and visible
plot(space(x,y, direction='y'))
title("Plot with 'space', direction='y' ")


# 'sunflower' plot, another approach, overlapping points are
# indicated via petals
sunflowerplot(x,y)
title("Sunflower Plot")



par(mfrow=prepar)
```

textplot	*Display text information in a graphics plot.*

Description

This function displays text output in a graphics window. It is the equivalent of 'print' except that the output is displayed as a plot.

Usage

```
textplot(object, halign="center", valign="center", cex, ...)
## Default S3 method:
textplot(object, halign=c("center","left","right"),
         valign=c("center", "top", "bottom"), cex, ... )
## S3 method for class 'character'
textplot(object, halign = c("center", "left", "right"),
         valign = c("center", "top", "bottom"), cex, fixed.width=TRUE,
         cspace=1, lspace=1, mar=c(0, 0, 3, 0) + 0.1,
         tab.width = 8, ...)
## S3 method for class 'data.frame'
textplot(object, halign = c("center", "left", "right"),
         valign = c("center", "top", "bottom"), cex, ...)
## S3 method for class 'matrix'
textplot(object, halign = c("center", "left", "right"),
         valign = c("center", "top", "bottom"), cex, cmar = 2,
         rmar = 0.5, show.rownames = TRUE, show.colnames = TRUE,
         hadj = 1, vadj = 1, mar = c(1, 1, 4, 1) + 0.1,
         col.data = par("col"), col.rownames = par("col"),
         col.colnames = par("col"), ...)
```

Arguments

`object`	Object to be displayed.
`halign`	Alignment in the x direction, one of "center", "left", or "right".
`valign`	Alignment in the y direction, one of "center", "top" , or "bottom"
`cex`	Character size, see `par` for details. If unset, the code will attempt to use the largest value which allows the entire object to be displayed.
`fixed.width`	Logical value indicating whether to emulate a fixed-width font by aligning characters in each row of text. This is usually necessary for text-formatted tables display properly. Defaults to 'TRUE'.
`cspace`	Space between characters as a multiple of the width of the letter 'W'. This only applies when `fixed.width==TRUE`.
`lspace`	Line spacing. This only applies when `fixed.width==TRUE`.
`mar`	Figure margins, see the documentation for `par`.
`rmar`, `cmar`	Space between rows or columns, in fractions of the size of the letter 'M'.

`show.rownames`, `show.colnames`	Logical value indicating whether row or column names will be displayed.
`hadj`, `vadj`	Vertical and horizontal location of elements within matrix cells. These have the same meaning as the `adj` graphics paramter (see `par`).
`col.data`	Colors for data elements. If a single value is provided, all data elements will be the same color. If a matrix matching the dimensions of the data is provided, each data element will receive the specified color.
`col.rownames`, `col.colnames`	Colors for row names and column names, respectively. Either may be specified as a scalar or a vector of appropriate length.
`tab.width`	Width of a single tab stop, in characters
`...`	Optional arguments passed to the text plotting command or specialied object methods

Details

A new plot is created and the object is displayed using the largest font that will fit on in the plotting region. The `halign` and `valign` parameters can be used to control the location of the string within the plotting region.

For matrixes and vectors a specialized textplot function is available, which plots each of the cells individually, with column widths set according to the sizes of the column elements. If present, row and column labels will be displayed in a bold font.

Value

The character scaling factor (`cex`) used.

Author(s)

Gregory R. Warnes <`greg@warnes.net`>

See Also

`plot`, `text`, `capture.output`

Examples

```
## Not run:
### simple examples
# show R version information
textplot(version)

# show the alphabet as a single string
textplot( paste(letters[1:26], collapse=" ") )

# show the alphabet as a matrix
textplot( matrix(letters[1:26], ncol=2))

### Make a nice 4 way display with two plots and two text summaries
```

```
data(iris)
par(mfrow=c(2,2))
plot( Sepal.Length ~ Species, data=iris, border="blue", col="cyan",
      main="Boxplot of Sepal Length by Species" )
plotmeans( Sepal.Length ~ Species, data=iris, barwidth=2, connect=FALSE,
           main="Means and 95% Confidence Intervals\nof Sepal Length by Species")

info <- sapply( split(iris$Sepal.Length, iris$Species),
                function(x) round(c(Mean=mean(x), SD=sd(x), N=nrow(x)),2) )

textplot( info, valign="top"  )
title("Sepal Length by Species")

reg <- lm( Sepal.Length ~ Species, data=iris )
textplot( capture.output(summary(reg)), valign="top")
title("Regression of Sepal Length by Species")

par(mfrow=c(1,1))

### Show how to control text color
cols <- c("red", "green", "magenta", "forestgreen")
mat <- cbind(name=cols, t(col2rgb(cols)), hex=col2hex(cols))

textplot(mat,
         col.data=matrix(cols, nrow=length(cols), byrow=FALSE, ncol=5),
         )

### Show how to manually tune the character size
data(iris)
reg <- lm( Sepal.Length ~ Species, data=iris )
text <- capture.output(summary(reg))

# do the plot and capture the character size used
textplot(text, valign="top")

# see what size was used
cex

# now redo the plot at 80% size
textplot( text, valign="top", cex=cex*0.80)




## End(Not run)
```

venn *Plot a Venn diagram*

Description

Plot a Venn diagrams for up to 5 sets

Usage

```
venn(data, universe=NA, small=0.7, showSetLogicLabel=FALSE,
     simplify=FALSE, show.plot=TRUE, intersections=TRUE, names,
     ...)

## S3 method for class 'venn'
plot(x, y, ..., small=0.7, showSetLogicLabel=FALSE,
     simplify=FALSE)
```

Arguments

`data, x`	Either a list list containing vectors of names or indices of group intersections, or a data frame containing boolean indicators of group intersectionship (see below)
`universe`	Subset of valid name/index elements. Values ignore values in `data` not in this list will be ignored. Use `NA` to use all elements of `data` (the default).
`small`	Character scaling of the smallest group counts
`showSetLogicLabel`	Logical flag indicating whether the internal group label should be displayed
`simplify`	Logical flag indicating whether unobserved groups should be omitted.
`show.plot`	Logical flag indicating whether the plot should be displayed. If false, simply returns the group count matrix.
`intersections`	Logical flag indicating if the returned object should have the attribute "individuals.in.intersections" featuring for every set a list of individuals that are assigned to it.
`y`	Ignored
`...`	Optional graphical parameters.
`names`	Optional vector of group names.

Details

`data` should be either a named list of vectors containing character string names ("GeneAABBB", "GeneBBBCY", .., "GeneXXZZ") or indexes of group intersections (1, 2, .., N), or a data frame containing indicator variables (TRUE, FALSE, TRUE, ..) for group intersectionship. Group names will be taken from the component list element or column names.

Value

Invisibly returns an object of class "venn", containing:

- A matrix of all possible sets of groups, and the observed count of items belonging to each The fist column contains observed counts, subsequent columns contain 0-1 indicators of group intersectionship.
- If `intersections=TRUE`, the attribute `intersections` will be a list of vectors containing the names of the elements belonging to each subset.

Author(s)

Steffen Moeller, with cleanup and packaging by Gregory R. Warnes.

Examples

```
##
## Example using a list of item names belonging to the
## specified group.
##

## construct some fake gene names..
oneName <- function() paste(sample(LETTERS,5,replace=TRUE),collapse="")
geneNames <- replicate(1000, oneName())

##
GroupA <- sample(geneNames, 400, replace=FALSE)
GroupB <- sample(geneNames, 750, replace=FALSE)
GroupC <- sample(geneNames, 250, replace=FALSE)
GroupD <- sample(geneNames, 300, replace=FALSE)
input  <-list(GroupA,GroupB,GroupC,GroupD)
input

tmp <- venn(input)
attr(tmp, "intersections")

##
## Example using a list of item indexes belonging to the
## specified group.
##
GroupA.i <- which(geneNames %in% GroupA)
GroupB.i <- which(geneNames %in% GroupB)
GroupC.i <- which(geneNames %in% GroupC)
GroupD.i <- which(geneNames %in% GroupD)
input.i  <-list(A=GroupA.i,B=GroupB.i,C=GroupC.i,D=GroupD.i)
input.i

venn(input.i)

##
## Example using a data frame of indicator ('f'lag) columns
##
GroupA.f <- geneNames %in% GroupA
GroupB.f <- geneNames %in% GroupB
GroupC.f <- geneNames %in% GroupC
GroupD.f <- geneNames %in% GroupD
input.df <- data.frame(A=GroupA.f,B=GroupB.f,C=GroupC.f,D=GroupD.f)
head(input.df)
venn(input.df)

## smaller set to create empty groupings
small <- input.df[1:20,]
```

```
venn(small, simplify=FALSE) # with empty groupings
venn(small, simplify=TRUE)  # without empty groupings

## Capture group counts, but don't plot
tmp <- venn(input, show.plot=FALSE)
tmp

## Show internal binary group labels
venn(input, showSetLogicLabel=TRUE)

## Limit  universe
tmp <- venn(input, universe=geneNames[1:100])
tmp

##
## Example to determine which elements are in A and B but not in
## C and D using the 'intersections' attribute.
##
tmp <- venn(input, intersection=TRUE)
isect <- attr(tmp, "intersection")

# Look at all of the subsets
str(isect)

# Extract and combine the subsets of interest..
AandB <- unique(c(isect$A, isect$B, isect$'A:B'))

# and look at the results
str(AandB)

##
## The full set of elements of each intersection is provided in the
## "interesections" attribute.
##
a<-venn(list(1:5,3:8), show.plot=FALSE)
intersections<-attr(a,"intersections")
print(intersections)
# $A
# [1] "1" "2"
#
# $B
# [1] "6" "7" "8"
#
# $`A:B`
# [1] "3" "4" "5"
```

wapply *Compute the Value of a Function Over a Local Region Of An X-Y Plot*

Description

This function applies the specified function to the sets of y values that are defined by overlapping "windows" in the x-dimension. For example, setting `fun=mean` returns local means, while setting `fun=function(x) sqrt(var(x))` returns local estimates of the standard deviation.

Usage

```
wapply(x, y, fun=mean, method="range", width, n=50, drop.na=TRUE,
       pts, ...)
```

Arguments

`x`	vector of x values for (x,y) pairs
`y`	vector of y values for (x,y) pairs
`fun`	function to be applied
`method`	method of defining an x-neighborhood. One of "width","nobs","range", or "fraction". See details.
`width`	width of an x-neighborhood. See details.
`n`	Number of equally spaced points at which to compute local estimates. See details.
`drop.na`	should points which result in missing values `NA` be omitted from the return value. Defaults to true.
`pts`	x locations at which to compute the local mean when using the "width" or "range" methods. Ignored otherwise.
`...`	arguments to be passed to `fun`

Details

Two basic techniques are available for determining what points fall within the same x-neighborhood. The first technique uses a window with a fixed width in the x-dimension and is is selected by setting `method="width"` or `method="range"`. For `method="width"` the `width` argument is an absolute distance in the x-dimension. For `method="range"`, the width is expressed as a fraction of the x-range. In both cases, `pts` specifies the points at which evaluation of `fun` occurs. When `pts` is omitted, `n` x values equally spaced along the x range are used.

The second technique uses windows containing k neighboring points. The (x,y) pairs are sorted by the x-values and the nearest k/2 points with higher x values and the k/2 nearest points with lower x values are included in the window. When `method="nobs"`, k equals `width` (actually 2*floor(`width`/2)). When `method="fraction"`, `width` specifies what fraction of the total number of points should be included. The actual number of points included in each window will be floor(n*frac/2)*2. Regardless of the value of `pts`, the function `fun` will be evaluated at all x locations.

Value

Returns a list with components

`x`	x location'
`y`	Result of applying `fun` to the window about each x location

Author(s)

Gregory R. Warnes <greg@warnes.net>

Examples

```
#show local mean and inner 2-sd interval to help diagnose changing mean
#or variance structure
x <- 1:1000
y <- rnorm(1000, mean=1, sd=1 + x/1000 )

plot(x,y)
lines(wapply(x,y,mean),col="red")

CL <- function(x,sd) mean(x)+sd*sqrt(var(x))

lines(wapply(x,y,CL,sd= 1),col="blue")
lines(wapply(x,y,CL,sd=-1),col="blue")
lines(wapply(x,y,CL,sd= 2),col="green")
lines(wapply(x,y,CL,sd=-2),col="green")

#show local mean and inner 2-sd interval to help diagnose changing mean
#or variance structure
x <- 1:1000
y <- rnorm(1000, mean=x/1000, sd=1)

plot(x,y)
lines(wapply(x,y,mean),col="red")

CL <- function(x,sd) mean(x)+sd*sqrt(var(x))

lines(wapply(x,y,CL,sd= 1,method="fraction",width=1/20),col="blue")
lines(wapply(x,y,CL,sd=-1,method="fraction",width=1/20),col="blue")
lines(wapply(x,y,CL,sd= 2,method="nobs",width=250),col="green")
lines(wapply(x,y,CL,sd=-2,method="nobs",width=250),col="green")
```

Index

∗ **aplot**
    angleAxis, 4
∗ **color**
    col2hex, 25
    colorpanel, 26
    rich.colors, 59
∗ **datasets**
    rtPCR, 60
∗ **design**
    qqnorm.aov, 56
∗ **dplot**
    balloonplot, 5
    bandplot, 9
    ci2d, 20
    hist2d, 39
    space, 63
    wapply, 70
∗ **hplot**
    balloonplot, 5
    barplot2, 11
    boxplot2, 15
    ci2d, 20
    heatmap.2, 29
    hist2d, 39
    lmplot2, 41
    ooplot.default, 45
    overplot, 49
    plotCI, 51
    plotmeans, 53
    qqnorm.aov, 56
    sinkplot, 62
    textplot, 65
    venn, 67
∗ **manip**
    reorder.factor, 57
∗ **misc**
    gplots-defunct, 27
    gplots-deprecated, 29
    residplot, 58
∗ **nonparametric**
    ci2d, 20
∗ **smooth**
    lowess, 43

adjust_hsv, 3
angleAxis, 4
axis, *4*, *14*, *47*

balloonplot, 5, *18*
bandplot, 9
barplot2, 11
bkde2D, *22*
bluered *(*colorpanel*)*, 26
boxplot, *16*, *48*, *55*
boxplot.n *(*gplots-defunct*)*, 27
boxplot2, 15, *28*
bubbleplot, *8*, 17, *19*

capture.output, *62*, *66*
catch.d, 19
catch.r *(*catch.d*)*, 19
ci2d, 20
col2hex, 25
col2rgb, *3*, *26*
colorpanel, 26
colors, *26*, *27*
conf2d, *22*
coplot, *50*

Defunct, *28*
dendrogram, *31*
Deprecated, *29*
deprecated *(*gplots-deprecated*)*, 29
dist, *32*
dotchart, *14*
dpik, *22*

factor, *58*
freq2d, *40*

gplots-defunct, 27
gplots-deprecated, 29
greenred *(*colorpanel*)*, 26

hclust, *32*, *35*
heat.colors, *60*
heatmap, *29*
heatmap.2, 29
hist, *14*, *40*
hist2d, *22*, 39
hsv, *3*

image, *29*, *34*, *35*, *40*
interaction.plot, *50*

jitter, *64*

layout, *34*, *35*
legend, *28*
lmplot2, *28*, 41
loess, *44*
lowess, *11*, 43

mixedsort, *58*

NULL, *34*

ooplot *(*ooplot.default*)*, 45
ooplot.default, 45
order.dendrogram, *34*
ordered, *57*
overplot, 49

panel.overplot *(*overplot*)*, 49
par, *14*, *33*, *35*, *47*, *65*, *66*
persp, *40*
plot, *14*, *48*, *50*, *66*
plot.lm, *42*
plot.lm2 *(*gplots-defunct*)*, 27
plot.lowess *(*lowess*)*, 43
plot.table, *8*
plot.venn *(*venn*)*, 67
plot.window, *14*, *47*
plotCI, 51, *55*
plotLowess *(*lowess*)*, 43
plotmeans, *53*, 53
points, *18*, *52*
print.ci2d *(*ci2d*)*, 20
print.hist2d *(*hist2d*)*, 39

qqnorm.aov, 56

rainbow, *60*
redblue *(*colorpanel*)*, 26
redgreen *(*colorpanel*)*, 26
reorder, *34*, *58*
reorder.factor, 57
residplot, 58
rev, *32*
rgb, *26*, *60*
rgb2hsv, *3*
rich.colors, 59
rtPCR, 60

sinkplot, 62
smartlegend *(*gplots-defunct*)*, 27
space, 63
stats, *32*
sunflowerplot, *63*, *64*
symbols, *18*

text, *16*, *66*
textplot, *62*, 65
title, *14*, *47*

venn, 67

wapply, *10*, *11*, 70