Package ‘plotfunctions’

October 14, 2022

Title Various Functions to Facilitate Visualization of Data and Analysis

Version 1.4

Date 2020-04-30

Author Jacolien van Rij [aut, cre]

Maintainer Jacolien van Rij <vanrij.jacolien@gmail.com>

Description When analyzing data, plots are a helpful tool for visualizing data and interpreting statistical models. This package provides a set of simple tools for building plots incrementally, starting with an empty plot region, and adding bars, data points, regression lines, error bars, gradient legends, density distributions in the margins, and even pictures. The package builds further on R graphics by simply combining functions and settings in order to reduce the amount of code to produce for the user. As a result, the package does not use formula input or special syntax, but can be used in combination with default R plot functions. Note: Most of the functions were part of the package 'itsadug', which is now split in two packages: 1. the package 'itsadug', which contains the core functions for visualizing and evaluating nonlinear regression models, and 2. the package 'plotfunctions', which contains more general plot functions.

URL https://jacolienvanrij.com/tutorials.html

License GPL (>= 2)

LazyData true

Depends R (>= 2.10)

VignetteBuilder knitr

Suggests knitr, sp

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2020-04-28 10:00:02 UTC

R topics documented:

addInterval *Draw intervals or arrows on plots.*

Description

Add horizontal or vertical interval indications. This function can also be used to plot asymmetric (non-parametric) error bars or confidence intervals. Basically a wrapper around arrows.

Usage

```
addInterval(
  pos,
  lowVals,
  highVals,
  horiz = TRUE,
  minmax = NULL,
  length = 0.05,
  ...
)
```

Arguments

`pos`	Vector with x- or y-values (depending on `horizontal`).
`lowVals`	Vector with low values, .
`highVals`	Vector with errors or confidence bands.
`horiz`	Logical: whether or not to plot the intervals horizontally. Defaults to TRUE (horizontal intervals).
`minmax`	Optional argument, vector with two values indicating the minimum and maximum value for the error bars. If NULL (default) the error bars are not corrected.
`length`	Number, size of the edges in inches.
`...`	Optional graphical parameters (see `par`) to be forwarded to the function `arrows`.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
emptyPlot(1000,5, xlab='Time', ylab='Y')
# add interval indication for Time=200 to Time=750:
addInterval(1, 200, 750, lwd=2, col='red')

# zero-length intervals also should work:
addInterval(pos=521, lowVals=c(1.35, 1.5, 4.33), highVals=c(1.15,1.5, 4.05),
    horiz=FALSE, length=.1, lwd=4)

# combine with getCoords for consistent positions with different axes:
par(mfrow=c(2,2))
# 1st plot:
emptyPlot(1000,c(-1,5), h0=0)
addInterval(getCoords(.1,side=2), 200,800,
    col='red', lwd=2)
addInterval(getCoords(.5,side=1), 1,4, horiz=FALSE,
    col='blue', length=.15, angle=100, lwd=4)
abline(h=getCoords(.1, side=2), lty=3, col='red', xpd=TRUE)
abline(v=getCoords(.5, side=1), lty=3, col='blue', xpd=TRUE)
# 2nd plot:
emptyPlot(1000,c(-250, 120), h0=0)
addInterval(getCoords(.1,side=2), 750,1200,
    col='red', lwd=2, minmax=c(0,1000))
abline(h=getCoords(.1, side=2), lty=3, col='red', xpd=TRUE)
# 3rd plot:
emptyPlot(c(-50,50),c(20,120), h0=0)
addInterval(getCoords(.5,side=1), 80,120, horiz=FALSE,
    col='blue', code=2, length=.15, lwd=4, lend=1)
abline(v=getCoords(.5, side=1), lty=3, col='blue', xpd=TRUE)

# Alternative boxplots:
b <- boxplot(count ~ spray, data = InsectSprays, plot=FALSE)$stats
emptyPlot(c(1,6), range(b[c(1,5),]), h0=0)
addInterval(1:6, b[1,], b[5,], horiz=FALSE)
# no end lines:
addInterval(1:6, b[2,], b[4,], horiz=FALSE, lwd=8, length=0, lend=2)
# no error with zero-length intervals:
addInterval(1:6, b[3,], b[3,], horiz=FALSE, lwd=2, length=.1, lend=2)

# reset
par(mfrow=c(1,1))
```

---

add_bars *Adding bars to an existing plot.*

Description

Adding bars to an existing plot.

Usage

```
add_bars(x, y, y0 = NULL, width = 1, horiz = FALSE, ...)
```

Arguments

`x`	Numeric vector with x-positions of bars.
`y`	Numeric vector with height of the bars.
`y0`	Optional numeric value or vector with the onset(s) of the bars. When \codey0 is not specified, the lowest value of the y-axis is used.
`width`	Numeric value, determining the width of the bars in units of the x-axis.
`horiz`	Logical value: whether or not to plot horizontal bars. Defaults to FALSE.
`...`	Other arguments for plotting, see \code\link[graphics]par.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# hypothetical experiment:
adults   =  stats::rpois(100, lambda = 5)
children =  stats::rpois(100, lambda = 4)
newd <- data.frame(Adults = table( factor(adults, levels=0:15) ),
    Children = table( factor(children, levels=0:15) ) )
newd <- newd[,c(1,2,4)]
names(newd)[1] <- 'value'

# barplot of Adults:
b <- barplot(newd$Adults.Freq, beside=TRUE, names.arg=newd$value,
    border=NA, ylim=c(0,30))
# overlay Children measures:
add_bars(b, newd$Children.Freq, col='red', density=25, xpd=TRUE)

# variants:
b <- barplot(newd$Adults.Freq, beside=TRUE, names.arg=newd$value,
    border=NA, ylim=c(0,30))
add_bars(b+.1, newd$Children.Freq, width=.85, col=alpha('red'),
    border=NA, xpd=TRUE)

emptyPlot(c(-30,30), c(0,15), v0=0, ylab='Condition')
add_bars(-1*newd$Children.Freq, 0:15, y0=0, col=alpha('blue'),
    border='blue', horiz=TRUE)
```

```
add_bars(newd$Adults.Freq, 0:15, y0=0, col=alpha('red'),
    border='red', horiz=TRUE)
mtext(c('Children', 'Adults'), side=3, at=c(-15,15), line=1, cex=1.25,
    font=2)

# adding shadow:
b <- barplot(newd$Adults.Freq, beside=TRUE, names.arg=newd$value,
    width=.9,
    col='black', border=NA)
add_bars(b+.2, newd$Adults.Freq+.2, y0=.2, width=.9,
    col=alpha('black', f=.2), border=NA, xpd=TRUE)
```

---

add_n_points *Add groups of points to a plot*

Description

Add groups of points to a plot

Usage

```
add_n_points(x, y, n, horiz = TRUE, width = NULL, sep = NULL, plot = TRUE, ...)
```

Arguments

x	average X position of points to plot,
y	average Y position of points to plot,
n	number of points to plot (integer).
horiz	Logical: whether or not to plot the sequence of point in horizontal direction (x-axis). Defaults to TRUE, the points are plotted in horizontal direction.
width	Numeric value: width that sequence of points can take.
sep	Numeric value: separation between sequences of points. Separation reduces the width. If the value is smaller, the sequences take more space.
plot	Logical: whether or not to add the points to the plot. Defaults to true. If set to false, the x- and y-coordinates of the points are returned in a list.
...	Optional graphical parameters (see par).

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: addInterval(), add_bars(), alphaPalette(), alpha(), check_normaldist(), color_contour(), dotplot_error(), drawDevArrows(), emptyPlot(), errorBars(), fill_area(), getCoords(), getFigCoords(), getProps(), gradientLegend(), legend_margin(), marginDensityPlot(), plot_error(), plot_image(), plotsurface(), sortBoxplot()

Examples

```
s <- table(cars$speed)
d <- tapply(cars$dist, list(cars$speed), mean)

emptyPlot(range(as.numeric(names(s))), range(d),
    xlab='dist', ylab='mean speed')
add_n_points(as.numeric(names(s)), d, s, pch='*')

# decrease space between groups of points:
emptyPlot(range(as.numeric(names(s))), range(d),
    xlab='dist', ylab='mean speed')
add_n_points(as.numeric(names(s)), d, s, sep=0)

# decrease width of groups of points:
emptyPlot(range(as.numeric(names(s))), range(d),
    xlab='dist', ylab='mean speed')
add_n_points(as.numeric(names(s)), d, s, width=0.8)

# horizontal vs vertical:
emptyPlot(range(d),range(as.numeric(names(s))),
    ylab='dist', xlab='mean speed')
add_n_points(d, as.numeric(names(s)), s, horiz=FALSE)
```

`alpha` *Adjusting the transparency of colors.*

Description

Wrapper around `adjustcolor`.

Usage

```
alpha(x, f = 0.5)
```

Arguments

`x`	A color or a vector with color values.
`f`	A number for adjusting the transparency ranging from 0 (completely transparent) to 1 (not transparent).

Note

Does not always work for x11 panels.

See Also

Other Functions for plotting: addInterval(), add_bars(), add_n_points(), alphaPalette(), check_normaldist(), color_contour(), dotplot_error(), drawDevArrows(), emptyPlot(), errorBars(), fill_area(), getCoords(), getFigCoords(), getProps(), gradientLegend(), legend_margin(), marginDensityPlot(), plot_error(), plot_image(), plotsurface(), sortBoxplot()

Examples

```
emptyPlot(100,100, h=50, v=50)
rect(25,25,75,75, col=alpha('red',f=1))
rect(35,41,63,81, col=alpha(rgb(0,1,.5),f=.25),
   border=alpha(rgb(0,1,.5), f=.65), lwd=4)

emptyPlot(1,1, axes=FALSE, main='Tunnel of 11 squares')
center <- c(.75, .25)
mycol <- 'steelblue'
for(i in seq(0,1,by=.1)){
    rect(center[1]-center[1]*(1.1-i), center[2]-center[2]*(1.1-i),
        center[1]+(1-center[1])*(1.1-i), center[2]+(1-center[2])*(1.1-i),
        col=alpha(mycol, f=i), border=mycol, lty=1, lwd=.5, xpd=TRUE)
}
axis(1, at=center[1]-center[1]*(1.1-seq(0,1,by=.1)), labels=seq(0,1,by=.1))

# see alphaPalette for an elaboration of this example
```

alphaPalette *Manipulate the transparency in a palette.*

Description

Generate an color palette with changing transparency.

Usage

```
alphaPalette(x, f.seq, n = NULL)
```

Arguments

x	A vector with color values. Could be a single value specifying a single color palette, ranging in transparency values, or a vector with different colors.
f.seq	A vector with transparency values, ranging from 0 to 1.
n	Optional argument. A number specifying the number of colors in the palette. If n > 1, then N transparency values are generated ranging from the minimum of f.seq to the maximum of f.seq. n will only be used when the vector f.seq has two elements or more.

Value

A vector with color values.

Warning

On Linux x11 devices may not support transparency. In that case, a solution might be to write the plots immediately to a file using functions such as pdf, or png.

Author(s)

Jacolien van Rij

See Also

palette, colorRampPalette, adjustcolor, convertColor

Other Functions for plotting: addInterval(), add_bars(), add_n_points(), alpha(), check_normaldist(), color_contour(), dotplot_error(), drawDevArrows(), emptyPlot(), errorBars(), fill_area(), getCoords(), getFigCoords(), getProps(), gradientLegend(), legend_margin(), marginDensityPlot(), plot_error(), plot_image(), plotsurface(), sortBoxplot()

Examples

```
# a palette of 5 white transparent colors:
alphaPalette('white', f.seq=1:5/5)
# the same palette:
alphaPalette('white', f.seq=c(.2,1), n=5)
# a palette with 10 colors blue, yellow and red, that differ in transparency
alphaPalette(c('blue', 'yellow', 'red'), f.seq=c(0.1,.8), n=10)

emptyPlot(1,1, axes=FALSE, main='Tunnel of 11 squares')
mycol <- 'steelblue'
center <- c(.75, .25)
i = seq(0,1,by=.1)
fillcol <- alphaPalette(c(mycol, 'black'), f.seq=i)
linecol <- alphaPalette(mycol, f.seq=1-i)
rect(center[1]-center[1]*(1.1-i), center[2]-center[2]*(1.1-i),
    center[1]+(1-center[1])*(1.1-i), center[2]+(1-center[2])*(1.1-i),
    col=fillcol, border=linecol, lty=1, lwd=1, xpd=TRUE)
```

`check_normaldist` *Compare distribution of data with normal distribution.*

Description

Compare distribution of data with normal distribution.

Usage

```
check_normaldist(
  res,
  col = "red",
  col.normal = "black",
  legend.pos = "topright",
  legend.label = "data",
  ...
)
```

Arguments

`res`	Vector with residuals or other data for which the distribution .
`col`	Color for filling the area. Default is black.
`col.normal`	Color for shading and line of normal distribution.
`legend.pos`	Position of legend, can be string (e.g., 'topleft') or an `xy.coords` object.
`legend.label`	Text string, label for plotted data distribution.
`...`	Optional arguments for the lines. See `par`.

Note

Assumes centered data as input.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
set.seed(123)
# normal distribution:
test <- rnorm(1000)
check_normaldist(test)
# t-distribution:
test <- rt(1000, df=5)
check_normaldist(test)
# skewed data, e.g., reaction times:
test <- exp(rnorm(1000, mean=.500, sd=.25))
check_normaldist(test)
# center first:
check_normaldist(scale(test))
# binomial distribution:
```

```
test <- rbinom(1000, 1, .3)
check_normaldist(test)
# count data:
test <- rbinom(1000, 100, .3)
check_normaldist(test)
```

`color_contour` *Creates a contour plot with colored background.*

Description

This function is a wrapper around `image` and `contour`. See `vignette('plotfunctions')` for an example of how you could use `image` and `contour`.

Usage

```
color_contour(
  x = seq(0, 1, length.out = nrow(z)),
  y = seq(0, 1, length.out = ncol(z)),
  z,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  col = NULL,
  color = topo.colors(50),
  nCol = 50,
  add.color.legend = TRUE,
  ...
)
```

Arguments

`x`	Locations of grid lines at which the values in z are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z.
`y`	Locations of grid lines at which the values in z are measured.
`z`	a matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience.
`main`	Text string, an overall title for the plot.
`xlab`	Label for x axis. Default is name of first `view` variable.
`ylab`	Label for y axis. Default is name of second `view` variable.
`xlim`	x-limits for the plot.

`ylim`	y-limits for the plot.
`zlim`	z-limits for the plot.
`col`	Color for the contour lines and labels.
`color`	a list of colors such as that generated by `rainbow`, `heat.colors` `colors`, `topo.colors`, `terrain.colors` or similar functions.
`nCol`	The number of colors to use in color schemes.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`...`	Optional parameters for `image` and `contour`.

Author(s)

Jacolien van Rij

See Also

`image`, `contour`, `filled.contour`. See `plotsurface` for plotting model predictions using `color_contour`.

`plotsurface`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# Volcano example of R (package datasets)
color_contour(z=volcano)
# change color and lines:
color_contour(z=volcano, color='terrain', col=alpha(1), lwd=2, lty=5)
# change x-axis values and zlim:
color_contour(x=seq(500,700, length=nrow(volcano)),
    z=volcano, color='terrain', col=alpha(1), lwd=2, zlim=c(0,200))

# compare with similar functions:
filled.contour(volcano, color.palette=terrain.colors)

# without contour lines:
color_contour(z=volcano, color='terrain', lwd=0, drawlabels=FALSE)
# without background:
color_contour(z=volcano, color=NULL, add.color.legend=FALSE)
```

`convertFile`	*Replacing separators (for example, decimal and thousand separators).*

Description

Replacing separators (for example, decimal and thousand separators).

Usage

```
convertFile(
  filename,
  symbol1 = NULL,
  symbol2 = NULL,
  newsymbol1 = "",
  newsymbol2 = "",
  sep = ";",
  newsep = NULL,
  header = TRUE,
  columns = NULL,
  outputfile = gsub("^(.*)(\\.)([^\\.]*)$", "\\1_new.\\3", filename),
  fixed.s1 = TRUE,
  fixed.s2 = TRUE,
  fixed.sep = TRUE,
  ...
)
```

Arguments

`filename`	String: filename (including path if necessary) of input file.
`symbol1`	String: symbol to replace by `newsymbol1`, for example decimal separator.
`symbol2`	String: second symbol to replace by `newsymbol2`, for example thousand separator.
`newsymbol1`	String: symbol to replace `symbol1`.
`newsymbol2`	String: symbol to replace `symbol2`.
`sep`	String: column separator. Could be also used to replace symbols in the header and data by `newsep`, regardless of columns.
`newsep`	String: symbol to replace `sep`. Only possible when `columns` is set to NULL.
`header`	Logical: whether or not there is header line. `symbol1` and `symbol2` are not replaced in the header line. Default set to TRUE.
`columns`	Vector with numerical values: indices of columns in which symbols need to be replaced.
`outputfile`	String: name of outputfile.
`fixed.s1`	Logical: whether or not to treat `symbol1` as fixed text instead of regular expression. Default is set to TRUE (no regular expression).

`fixed.s2`	Logical: whether or not to treat `symbol2` as fixed text instead of regular expression. Default is set to TRUE (no regular expression).
`fixed.sep`	Logical: whether or not to treat `sep` as fixed text instead of regular expression. Default is set to TRUE (no regular expression).
`...`	Additional parameters for `read.table` and `write.table`.

Author(s)

Jacolien van Rij

Examples

```
## Not run:
# normally, the function call would look something like this:
convertFile('example1.csv', symbol1=',', symbol2='.', sep='\t',
    newsymbol1='.', newsymbol2='')
# But as we are not sure that the file example1.csv is available,
# we need to do something a little more complicated to point to
# the file 'example1.csv' that comes with the package:

# finding one of the example files from the package:
file1 <- system.file('extdata', 'example1.csv', package = 'plotfunctions')

# example 1:
system.time({
    convertFile(file1, symbol1=',', symbol2='.',
    newsymbol1='.', newsymbol2='', outputfile='example1_new.csv')
})
# example 2: type 'yes' to overwrite the previous output file,
# or specify a different filename in outputfile.
system.time({
    convertFile(file1, symbol1=',', symbol2='.', sep='\t',
    newsymbol1='.', newsymbol2='', columns=1:2, outputfile='example1_new.csv')
})
# Example 1 takes less  time, as it does not use read.table,
# but just reads the file as text lines. However, the column
# version could be useful when symbols should be replaced only
# in specific columns.
# Note that Example 2 writes the output with quotes, but this is
# not a problem for read.table:
dat <- read.table('example1_new.csv', header=TRUE, sep='\t',
    stringsAsFactors=FALSE)

## End(Not run)
```

`dotplot_error` *Utility function*

Description

Adjusted version of the a Cleveland dot plot implemented in dotchart with the option to add confidence intervals.

Usage

```
dotplot_error(
  x,
  se.val = NULL,
  labels = NULL,
  groups = NULL,
  gdata = NULL,
  cex = par("cex"),
  pch = 21,
  gpch = 21,
  bg = "black",
  color = par("fg"),
  gcolor = par("fg"),
  lcolor = "gray",
  xlim = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  lwd = 1,
  ...
)
```

Arguments

`x`	either a vector or matrix of numeric values (NAs are allowed). If x is a matrix the overall plot consists of juxtaposed dotplots for each row. Inputs which satisfy is.numeric(x) but not is.vector(x) || is.matrix(x) are coerced by as.numeric, with a warning.
`se.val`	a vector or matrix of numeric values representing the standard error or confidence bands.
`labels`	a vector of labels for each point. For vectors the default is to use names(x) and for matrices the row labels dimnames(x)[[1]].
`groups`	an optional factor indicating how the elements of x are grouped. If x is a matrix, groups will default to the columns of x.
`gdata`	data values for the groups. This is typically a summary such as the median or mean of each group.
`cex`	the character size to be used. Setting cex to a value smaller than one can be a useful way of avoiding label overlap. Unlike many other graphics functions, this sets the actual size, not a multiple of par('cex').
`pch`	the plotting character or symbol to be used.
`gpch`	the plotting character or symbol to be used for group values.

`bg`	the background color of plotting characters or symbols to be used; use par(bg= *) to set the background color of the whole plot.
`color`	the color(s) to be used for points and labels.
`gcolor`	the single color to be used for group labels and values.
`lcolor`	the color(s) to be used for the horizontal lines.
`xlim`	horizontal range for the plot, see plot.window, e.g.
`main`	overall title for the plot, see title.
`xlab`	x-axis annotation as in title.
`ylab`	y-axis annotation as in title.
`lwd`	with of error bars.
`...`	graphical parameters can also be specified as arguments see `par`

Author(s)

This function is a slightly adjusted version of the function `dotchart` of the package `graphics` version 3.1.1

See Also

`dotchart`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# example InsectSprays from R datasets
avg <- aggregate(count ~ spray, data=InsectSprays, mean)
avg <- merge(avg,
    aggregate(count ~ spray, data=InsectSprays, sd),
    by='spray', all=TRUE)

dotplot_error(avg$count.x, se.val=avg$count.y, labels=avg$spray)

# we could add the type of spray to the averages:
avg$type <- c(1,1,2,2,2,1)
dotplot_error(avg$count.x, se.val=avg$count.y, groups=avg$type, labels=avg$spray)
```

drawDevArrows — *Draw arrows between different plots.*

Description

Function for drawing arrows between different plot regions.

Usage

```
drawDevArrows(
  start,
  end = NULL,
  arrows = c("end", "start", "both", "none"),
  units = c("inch", "prop", "coords"),
  ...
)
```

Arguments

`start`	The x and y coordinates of a set of points that define the start points of the arrow(s), specified in a list with x and y slots. Similar to `xy.coords`. Alternatively, start could be a matrix with start and end points in two columns. In that case, end is set to NULL.
`end`	The x and y coordinates of a set of points that define the end points of the arrow(s), specified in a list with x and y slots.
`arrows`	On which end of the line to draw arrows: 'end' (default), 'start', 'both', 'none'.
`units`	Units in which x- and y-coordinates are provided: 'inch' (default), 'prop' (proportion), 'coords'. 'inch' and 'prop' are with respect to device region.
`...`	graphical parameters and parameters provided for `arrows`.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
### EXAMPLE 1 ################################

# setup 4 panels:
```

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

#------------------
# PLOT 1: two points
#------------------

plot(0.5, 0.5, main='1',
    pch=21, lwd=3, col='red', bg='white', cex=1.2)
points(.5, .375, pch=22, lwd=3, col='blue', cex=1.2)

# Draw an error between the two points:
drawDevArrows(start=c(.5,.5), end=c(.5,.375),
    units='coords', arrows='start', length=.1, lty=1)
# ... which is the same as using arrows:
arrows(x0=.5, x1=.5, y0=.5, y1=.375, code=1, length=.1, lty=1)

# ... but these arrows can also be clipped to the device
# instead of the plot region (see leftbottom corner):
drawDevArrows(start=c(.5,.5), end=c(.5,.375),
    units='dev', arrows='start', length=.1, lty=1)

# The function getArrowPos converts coordinates to device coordinates:
x1 <- getArrowPos(x=0.5, y=0.5, units='coords')
x2 <- getArrowPos(x=0.5, y=0.375, units='coords')
drawDevArrows(x1, x2, col='purple',
    arrows='start', length=.1, lty=2, lwd=2)


# Setup 4 arrows with the same starting points,
# but defined differently:
a1 <- getArrowPos(x=0.5, y=0.375, units='coords')
a2 <- getArrowPos(x=0.5, y=0.21, units='prop')
a3 <- getArrowPos(x=0.55, y=0.36, units='prop', dev='fig')
a4 <- getArrowPos(x=0.5*0.55, y=.5*0.36+.5, units='prop', dev='dev')

# Setup 3 arrows with the same x and y values,
# which define different starting points in practice:
b1 <- getArrowPos(x=.5, y=.5, units='prop', dev='plot')
b2 <- getArrowPos(x=.5, y=.5, units='prop', dev='fig')
b3 <- getArrowPos(x=.5, y=.5, units='prop', dev='dev')


#------------------
# PLOT 2: different coordinates
#------------------

plot(c(-2.33, 20), c(.3, .8), type='n', main='2')
points(15,.8, pch=21, lwd=3, col='red', bg='white', cex=1.2)

# define end point for b:
b <- getArrowPos(x=15, y=.8)

# Draw arrow b1:
```

```
drawDevArrows(start=b1, end=b, arrows='start', length=.1, lty=1)


#------------------
# PLOT 3: upside down axis
#------------------

emptyPlot(c(25, 1050), c(15,-15), eegAxis=TRUE, h0=0)
# plot line:
x <- 0:1000
y <- 10*cos(x/100)
lines(x, y, col=4)
# draw point points on gthe line:
x <- c(200,400,600,800)
y <- 10*cos(x/100)
points(x,y, pch=18)

# To avoid calling the function drawDevArrows 4 times, we rewrite
# the x- and y-positions of the 4 coordinates a1, a2, a3, a4 in one list:
a.start <- list(x=c(a1$x, a2$x, a3$x, a4$x), y=c(a1$y, a2$y, a3$y, a4$y))
# Define end points on the line:
a.end <- getArrowPos(x=x, y=y)
drawDevArrows(start=a.start, end=a.end, arrows='none', lty=3)

# Note that these four coordinates are actually referring
# to the same starting point!
# So instead we could have written:
drawDevArrows(start=a1, end=a.end, arrows='none', col=alpha('red'), lwd=2)


#------------------
# PLOT 4: wrapping up
#------------------

# Arrows could be constructed when the plot is not yet called,
# as they are clipped to the device:
drawDevArrows(start=c(0,7), end=c(7,0), col='gray', lwd=4, lty=3, arrows='none')

# Add the plot:
plot(1,1, bg='green')

# Finish b2 and b3: same x and y, but different coordinates
drawDevArrows(start=b2, end=b, arrows='start', length=.1, lty=2)
drawDevArrows(start=b3, end=b, arrows='start', length=.1, lty=3)



### EXAMPLE 2 ################################



# setup 4 plots:
par(mfrow=c(2,2))
```

```
n <- 50

#------------------
# PLOT 1: empty
#------------------


emptyPlot(c(25, 1050), c(15,-15), axes=FALSE)
lines(0:1000, 10*cos(0:1000/200), col=4)
x <- seq(0,1000, length=n)
y <- 10*cos(x/200)

a <- getArrowPos(x=x, y=y)


#------------------
# PLOT 2
#------------------

emptyPlot(c(25, 1050), c(15,-15), axes=FALSE)
lines(0:1000, 10*sin(0:1000/200), col=1)
x <- seq(0,1000, length=n)
y <- 10*sin(x/200)


b <- getArrowPos(x=x, y=y)



#------------------
# PLOT 3
#------------------

emptyPlot(c(25, 1050), c(15,-15), axes=FALSE)
lines(0:1000, 10*cos(0:1000/200), col=4)
x <- seq(0,1000, length=n)
y <- 10*cos(x/200)


c <- getArrowPos(x=rev(x), y=rev(y))


#------------------
# PLOT 4
#------------------

emptyPlot(c(25, 1050), c(15,-15), axes=FALSE)
lines(0:1000, 10*sin(0:1000/200), col=1)
x <- seq(0,1000, length=n)
y <- 10*sin(x/200)

d1 <- getArrowPos(x=rev(x), y=rev(y))
```

```
d2 <- getArrowPos(x=x, y=y)


#------------------
# DRAW ARROWS
#------------------

drawDevArrows(start=a, end=b, arrows='none', col='gray')
drawDevArrows(start=c, end=d1, arrows='none', col='gray')

drawDevArrows(start=a, end=c, arrows='none',
    col=alphaPalette(c('green', 'blue'), f.seq=c(0,1), n=n))
drawDevArrows(start=b, end=d2, arrows='none',
    col=alphaPalette('pink', f.seq=c(1,.1), n=n))
```

---

emptyPlot *Utility function*

Description

Generate an empty plot window.

Usage

```
emptyPlot(
  xlim,
  ylim,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  h0 = NULL,
  v0 = NULL,
  bty = "n",
  eegAxis = FALSE,
  xmark = NULL,
  ymark = NULL,
  ...
)
```

Arguments

xlim	A one- or two-value vector indicating the range of the x-axis. If `xlim` is a number, then it is assumed that the other value is 0. Thus, `xlim=3000` wil result in a x-axis ranging from 0 to 3000, and `xlim=-3` will result in a x-axis ranging from -3 to 0. Optional, `xlim` can be a vector with categorical x-axis labels.
ylim	A one- or two-value vector indicating the range of the y-axis. (See `xlim`) for more information. Optional, `ylim` can be a vector with categorical y-axis labels.

`main`	Title for the plot. Empty by default. Note: the title can be added later using `title`.
`xlab`	Label for x-axis. Empty by default. If no label is provided, use `mtext` for adding axis labels.
`ylab`	Label for y-axis. Empty by default. (See `xlab`.)
`h0`	A vector indicating where to add solid horizontal lines for reference. By default no values provided.
`v0`	A vector indicating where to add dotted vertical lines for reference. By default no values provided.
`bty`	A character string which determined the type of box which is drawn about plots. If bty is one of 'o', 'l', '7', 'c', 'u', or ']' the resulting box resembles the corresponding upper case letter. A value of 'n' (the default) suppresses the box.
`eegAxis`	Logical: whether or not to reverse the y-axis, plotting the negative amplitudes upwards as traditionally is done in EEG research. If eeg.axes is TRUE, labels for x- and y-axis are provided, when not provided by the user. Default value is FALSE.
`xmark`	Numeric factor with x-axis tick marks and limits. If NULL (default) R's default system is used. If TRUE, only the `xlim` values are indicated.
`ymark`	Numeric factor with y-axis tick marks and limits. If NULL (default) R's default system is used. If TRUE, only the `ylim` values are indicated.
`...`	Other arguments for plotting, see `par`.

Value

An empty plot window.

Author(s)

Jacolien van Rij

See Also

Use `title` and `mtext` for drawing labels and titles; use `lines` and `points` for plotting the data; use `legend` or `legend_margin` for adding a legend.

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# generate some measurements:
x <- runif(100,0,100)
y <- rpois(100,lambda=3)

# Setup empty plot window fitting for data:
emptyPlot(range(x), range(y))
```

```
# To add data, use lines() and points()
points(x,y, pch=16, col=alpha('steelblue'))

# Category labels:
emptyPlot(toupper(letters[1:5]), 1)
# order matters:
emptyPlot(sample(toupper(letters[1:5])), 1)
# actually, they are plotted on x-positions 1:5
points(1:5, rnorm(5, mean=.5, sd=.1))
# also possible for y-axis or both:
emptyPlot(c(200,700), toupper(letters[1:5]))
emptyPlot(as.character(8:3), toupper(letters[1:5]))
# change orientation of labels:
par(las=1)
emptyPlot(c(200,700), toupper(letters[1:5]))
par(las=0) # set back to default

# More options:
emptyPlot(range(x), range(y),
    main='Data', ylab='Y', xlab='Time')
# add averages:
m <- tapply(y, list(round(x/10)*10), mean)
lines(as.numeric(names(m)), m, type='o', pch=4)

# with vertical and horizontal lines:
emptyPlot(1, 1, h0=.5, v0=.75)
# eeg axis (note the axes labels):
emptyPlot(c(-200,1000), c(-5,5),
    main='EEG', v0=0, h0=0,
    eegAxis=TRUE)

# simplify axes:
emptyPlot(c(-3.2,1.1), c(53,58),
    xmark=TRUE, ymark=TRUE, las=1)
# compare with R default:
emptyPlot(c(-3.2,1.1), c(53,58), las=1)
# also possible to specify values manually:
emptyPlot(c(-3.2,1.1), c(53,58),
    xmark=c(-3.2,0, 1.1), ymark=c(55,57), las=1)

# empty window:
emptyPlot(1,1,axes=FALSE)
# add box:
emptyPlot(1,1, bty='o')
```

errorBars *Add error bars to a plot.*

Description

Add vertical error bars.

Usage

```
errorBars(
  x,
  mean,
  ci,
  ci.l = NULL,
  minmax = NULL,
  horiz = FALSE,
  border = FALSE,
  ...
)
```

Arguments

`x`	Vector with x-values (or y-values in case `horiz=TRUE`).
`mean`	Vector with means.
`ci`	Vector with errors or confidence bands, e.g. SE values. If `ci.l` is not defined, the errors are assumed to be symmetric. If `ci.l` is defined, than `ci` is assumed to be the upper confidence band. Note that the `ci` will be added (or substracted) from the mean.
`ci.l`	Optional: vector with error to calculate lower confidence band.
`minmax`	Optional argument, vector with two values indicating the minimum and maximum value for the error bars. If NULL (default) the error bars are not corrected.
`horiz`	Logical: whether or not to plot horizontal error bars. Defaults to FALSE (plotting vertical error bars).
`border`	Logical: whether or not to add a border around the error bars. Defaults to FALSE (no border added). Color and width of the borders can be adjusted using `border.col` and `border.width`. (Basically all parameters that work for errorBars, with `border.` added before.) See examples.
`...`	Optional graphical parameters (see `par`).

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# example InsectSprays from R datasets
```

```
InsectSprays$type <- ifelse( InsectSprays$spray %in% c('A', 'B', 'F'), 1,2)
avg <- with(InsectSprays, tapply(count, list(spray), mean))
sds <- with(InsectSprays, tapply(count, list(spray), sd))


# barplot:
b <- barplot(avg, beside=TRUE, main='Insect Sprays', ylim=c(0,20))
errorBars(b, avg, sds, xpd=TRUE, length=.05)

# constrain error bars to max and min of plot:
b <- barplot(avg, beside=TRUE, main='Insect Sprays', ylim=c(0,20))
errorBars(b, avg, sds, minmax=c(0,20), xpd=TRUE, length=.05)

# add borders:
b <- barplot(avg, beside=TRUE, main='Insect Sprays', ylim=c(0,20),
     col=1, border=NA)
errorBars(b, avg, sds, minmax=c(0,20), xpd=TRUE, length=.05, border=TRUE)

# change layout:
b <- barplot(avg, beside=TRUE, main='Insect Sprays', ylim=c(0,20),
     col=1, border=NA)
errorBars(b, avg, sds, minmax=c(0,20), xpd=TRUE, border=TRUE,
     length=.05, col='blue', # settings for error bars
     border.length=.1, border.col='yellow', border.lwd=5) # settings border

# line plot with asymmetric fake errors:
emptyPlot(toupper(letters[1:6]), 20, main='Averages', xlab='Spray')
ci.low <- abs(rnorm(6, mean=2))
ci.high <-  abs(rnorm(6, mean=4))

errorBars(1:6, avg, ci.high, ci.l= ci.low, length=.05, lwd=2)
points(1:6, avg, pch=21, type='o', lty=3, lwd=2,
    bg='white', xpd=TRUE)
# also horizontal bars possible:
errorBars(10, 1, 1.2, horiz=TRUE, col='red')
```

fill_area *Utility function*

Description

Fill area under line or plot.

Usage

```
fill_area(
  x,
  y,
  from = 0,
  col = "black",
  alpha = 0.25,
  border = NA,
  na.rm = TRUE,
  horiz = TRUE,
  outline = FALSE,
  ...
)
```

Arguments

`x`	Vector with values on x-axis.
`y`	Vector with values on y-axis.
`from`	A number indicating until which value on the y-axis the graph is colored. Defaults to 0.
`col`	Color for filling the area. Default is black.
`alpha`	Transparency of shaded area. Number between 0 (completely transparent) and 1 (not transparent). Default is .25.
`border`	A color, indicating the color of the border around shaded area. No border with value NA (default).
`na.rm`	Logical: whether or not to remove the missing values in `x` and `y`. Defaults to TRUE. If set to FALSE, missing values may cause that the filled area is split in various smaller areas.
`horiz`	Logical: whether or not to plot with respect to the x-axis (TRUE) or y-xis (FALSE). Defaults to TRUE.
`outline`	Logical: whether or not to draw the outline instead of only the upper border of the shape. Default is FALSE (no complete outline).
`...`	Optional arguments for the lines. See `par`.

Author(s)

Jacolien van Rij

See Also

`check_normaldist`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# density of a random sample from normal distribution:
test <- density(rnorm(1000))
emptyPlot(range(test$x), range(test$y))
fill_area(test$x, test$y)
fill_area(test$x, test$y, from=.1, col='red')
fill_area(test$x, test$y, from=.2, col='blue', density=10, lwd=3)
lines(test$x, test$y, lwd=2)
```

`findAbsMin` *Return the value (or the element with the value) closest to zero.*

Description

Return the value (or the element with the value) closest to zero.

Usage

```
findAbsMin(x, element = FALSE)
```

Arguments

`x`	A numeric vector.
`element`	Logical: whether or not to return the value (FALSE, default) or the index (TRUE).

Value

The value or index of the element closest to zero (absolute minimum).

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

Examples

```
(test <- seq(-25,25, by=3))
min(test[test>0])
max(test[test<0])
min(abs(test))
findAbsMin(test)
```

find_n_neighbors *Return n neighbors around given indices.*

Description

Return n neighbors around given indices.

Usage

```
find_n_neighbors(el, n, max)
```

Arguments

el	A numeric vector.
n	Number indicating how many points around the elements of el need to be selected.
max	The maximum value of the returned elements.

Value

A vector with the elements of x surrounded by n points.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: findAbsMin(), firstLetterCap(), getArrowPos(), getDec(), getRange(), getRatioCoords(), get_palette(), group_sort(), inch2coords(), isColor(), list2str(), move_n_point(), orderBoxplot(), se(), sortGroups()

Examples

```
vectorIndices <- 1:1000
indOutliers <- c(2,10, 473, 359, 717, 519)
fn3 <- find_n_neighbors(indOutliers, n=3, max=max(vectorIndices))
fn20 <- find_n_neighbors(indOutliers, n=20, max=max(vectorIndices))

# check fn3:
print(fn3)

# Plot:
emptyPlot(c(-10,1000), c(-1,1), h0=0, v0=indOutliers)
points(fn3, rep(.5, length(fn3)), pch='*')
points(fn20, rep(-.5, length(fn20)), pch='*')
```

`firstLetterCap` *Capitalize first letter of a string.*

Description

Capitalize first letter of a string.

Usage

```
firstLetterCap(x)
```

Arguments

`x`	Text string

Value

Text string

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

`getArrowPos` *Converts coordinates in current plot region to device positions (in inch).*

Description

Converts coordinates in current plot region to device positions (in inch).

Usage

```
getArrowPos(
  x,
  y,
  units = c("coords", "prop"),
  dev = c("plot", "figure", "panel")
)
```

Arguments

`x`	Numeric: x coordinate(s)
`y`	Numeric: y coordinate(s)
`units`	Coordinates (default) or proportions with respect to plot region.
`dev`	x and y position are measured with respect to the plot region (default), figure panel, or device.

Value

list

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

`getCoords` *Convert proportions into coordinates of the plot or figure region.*

Description

Function for positioning a legend or label in or outside the plot region based on proportion of the plot region rather than Cartesian coordinates.

Usage

```
getCoords(pos = 1.1, side = 1, input = "p")
```

Arguments

`pos`	A number indicating the proportion on the x-axis. Default is 1.1.
`side`	Which axis to choose: 1=bottom, 2=left, 3=top, 4=right. Default is 1.
`input`	Which proportion to take: with respect to the plot region (input 'p', default), or with respect to figure region (input 'f').

Author(s)

Jacolien van Rij

See Also

`getFigCoords`, `getProps`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# set larger plot window, depending on your system:
# dev.new(,with=8, height=4) # windows, mac
# quartz(,8,4)               # Mac
# x11(width=8, height=4)     # linux
par(mfrow=c(1,2))

# PLOT 1: y-range is -1 to 1
emptyPlot(c(0,1),c(-1,1), h0=0, v0=0.5)
# calculate the x-coordinates for points at proportion
# -0.2, 0, .25, .5, 1.0, and 1.1 of the plot window:
p1 <- getCoords(pos=c(-0.2,0,.25,.5,1,1.1), side=2)
# use xpd=TRUE to plot outside plot region:
points(rep(0.5,length(p1)), p1, pch=16, xpd=TRUE)
# add legend outside plot region, in upper-right corner of figure:
legend(x=getCoords(1,side=1, input='f'), y=getCoords(1, side=2, input='f'),
    xjust=1, yjust=1,
    legend=c('points'), pch=16, xpd=TRUE)
# Note: this can easier be achieved with function getFigCoords

# PLOT 2: y-range is 25 to 37
# we would like to plot the points and legend at same positions
emptyPlot(c(0,1),c(25,37), h0=0, v0=0.5)
p1 <- getCoords(pos=c(-0.2,0,.25,.5,1,1.1), side=2)
points(rep(0.5,length(p1)), p1, pch=16, xpd=TRUE)
# add legend outside plot region, in upper-left corner of figure:
legend(x=getCoords(0,side=1, input='f'), y=getCoords(1, side=2, input='f'),
    xjust=0, yjust=1,
    legend=c('points'), pch=16, xpd=TRUE)
```

`getDec` *Return the number of decimal places.*

Description

Return the number of decimal places.

Usage

```
getDec(x)
```

Arguments

`x`	A numeric vector.

Value

Number of decimals

Author(s)

Based on http://stackoverflow.com/questions/5173692/how-to-return-number-of-decimal-places-in-r, but improved

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

Examples

```
getDec(c(10,10.432, 11.01, .000001))
```

`getFigCoords` *Get the figure region as coordinates of the current plot region, or as corrdinates of the figure region.*

Description

Get the figure region as coordinates of the current plot region, or as corrdinates of the figure region.

Usage

```
getFigCoords(input = "f")
```

Arguments

`input`	Text string: 'f' (figure, default), 'p' (plot region), 'hf' (half way figure region), or 'hp' (half way plot region)

Value

A vector of the form c(x1, x2, y1, y2) giving the boundaries of the figure region as coordinates of the current plot region.

Author(s)

Jacolien van Rij

See Also

`getCoords`, `getProps`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# setup plot region:
emptyPlot(1,1, bty='o')
fc <- getFigCoords()
pc <- getFigCoords('p')
arrows(x0=pc[c(1,2,1,2)], x1=fc[c(1,2,1,2)],
    y0=pc[c(3,3,4,4)], y1=fc[c(3,3,4,4)], xpd=TRUE)

# Same plot with different axis:
emptyPlot(c(250,500),c(331, 336), bty='o')
fc <- getFigCoords()
pc <- getFigCoords('p')
arrows(x0=pc[c(1,2,1,2)], x1=fc[c(1,2,1,2)],
    y0=pc[c(3,3,4,4)], y1=fc[c(3,3,4,4)], xpd=TRUE)
hc <-  getFigCoords('h')

# other options:
# 1. center of figure region:
abline(v=getFigCoords('hf')[1], col='blue', xpd=TRUE)
abline(h=getFigCoords('hf')[2], col='blue', xpd=TRUE)
# 2. center of plot region:
abline(v=getFigCoords('hp')[1], col='red', lty=3)
abline(h=getFigCoords('hp')[2], col='red', lty=3)
```

`getProps` *Transform coordinates into proportions of the figure or plot region.*

Description

Function for positioning a legend or label in or outside the plot region based on proportion of the plot region rather than Cartesian coordinates.

Usage

```
getProps(pos, side = 1, output = "p")
```

Arguments

pos	A number indicating the coordinates on the x- or y-axis.
side	Which axis to choose: 1=bottom, 2=left, 3=top, 4=right. Default is 1.
output	Which proportion to take: with respect to the plot region (input 'p', default), or with respect to figure region (output 'f').

Author(s)

Jacolien van Rij

See Also

getCoords, getFigCoords

Other Functions for plotting: addInterval(), add_bars(), add_n_points(), alphaPalette(), alpha(), check_normaldist(), color_contour(), dotplot_error(), drawDevArrows(), emptyPlot(), errorBars(), fill_area(), getCoords(), getFigCoords(), gradientLegend(), legend_margin(), marginDensityPlot(), plot_error(), plot_image(), plotsurface(), sortBoxplot()

Examples

```
# not very easy-to-calculate-with x- and y-axis values
emptyPlot(c(-2.35, 37.4), c(9,11), v0=0)
# draw a mirror symmetric image of boxes:
p1 <- c(9.5, 9.5)
p2 <- c(4,9.7)
p3 <- c(20,9)
p1m <- getCoords(1-getProps(p1, side=c(1,2)), side=c(1,2))
p2m <- getCoords(1-getProps(p2, side=c(1,2)), side=c(1,2))
p3m <- getCoords(1-getProps(p3, side=c(1,2)), side=c(1,2))
xdist <- diff(getCoords(c(0,.1), side=1))
ydist <- diff(getCoords(c(0,.1), side=2))
rect(xleft=c(p1[1],p2[1], p3[1], p1m[1], p2m[1], p3m[1])-xdist,
    xright=c(p1[1],p2[1], p3[1], p1m[1], p2m[1], p3m[1])+xdist,
    ybottom=c(p1[2],p2[2], p3[2], p1m[2], p2m[2], p3m[2])-ydist,
    ytop=c(p1[2],p2[2], p3[2], p1m[2], p2m[2], p3m[2])+ydist,
    col=rep(c('red', NA, 'lightblue'),2), xpd=TRUE )
```

---

getRange *Function for rounding and/or segmenting a range.*

Description

Function for rounding and/or segmenting a range.

Usage

```
getRange(x, dec = NULL, step = NULL, n.seg = 2)
```

Arguments

x	A numeric vector.
dec	Number of decimal points for rounding using function round. Applied after argument step. If NULL (default), no rounding is applied.
step	Round the
n.seg	Numeric value, number of values in the equally spaced sequence. Default is 2 (min, max).

Value

vector, range of equally spaced sequence.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: findAbsMin(), find_n_neighbors(), firstLetterCap(), getArrowPos(), getDec(), getRatioCoords(), get_palette(), group_sort(), inch2coords(), isColor(), list2str(), move_n_point(), orderBoxplot(), se(), sortGroups()

Examples

```
zlim <- c(-2.5, 3.01)
# does not change anything:
getRange(zlim)
# create a range of 5 numbers:
# (basically just using seq )
getRange(zlim, n.seg=5)
# rounds the numbers:
getRange(zlim, dec=0)
getRange(zlim, n.seg=5, dec=0)
# extreme values are multiplications of 5
# that contains zlim values:
getRange(zlim, step=5)
getRange(zlim, step=5, n.seg=5)
# similar, but not the same:
getRange(zlim, n.seg=5, dec=0)
getRange(zlim, n.seg=5, step=1)
# combining:
getRange(zlim, n.seg=5, step=1, dec=0)
```

---

getRatioCoords *Move a vector n elements forward or backward.*

Description

Move a vector n elements forward or backward.

Usage

```
getRatioCoords(
  ratio,
  width = NULL,
  height = NULL,
  input = c("coords", "prop"),
  ...
)
```

Arguments

`ratio`	Numeric, height : width ratio. If `ratio` > 1, the width is larger than the height, if `ration` < 1, the height is larger than the width.
`width`	The desired width in plot coordinates or proportions. If not specified (NULL), the maximal width fitting in the plot region is returned.
`height`	The desired height in plot coordinates or proportions. If not specified (NULL), the maximal height fitting in the plot region is returned.
`input`	Unit of input width and height, 'coords' (plot coordinates, default), or 'prop' (proportions of plot region).
`...`	Optional arguments: `xcenter`, `xleft`, or `xright`, and `ycenter`, `ybottom`, or `ytop` could be specified. If not specified, the width and height are centered around the center of the plot.

Value

A list with 5 elements:

- `width`: width of the element in x-axis coordinates;
- `height`: height of the element in y-axis coordinates;
- `ratio`: provided ratio (for confirmation);
- `x`: two-number vector with x-coordinates of left and right sides;
- `y`: two-number vector with y-coordinates of bottom and top sides.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

Examples

```
data(img)
emptyPlot(100, c(50, 100), h0=0, v0=0)
# calculate height : width ratio of image:
im.r <- dim(img$image)[1]/dim(img$image)[2]
p <- getRatioCoords(ratio=im.r, width=20)
# inspect p:
p
# No position specified, so centered:
plot_image(img, type='image', add=TRUE,
    xrange=p$x, yrange=p$y)
# ... or we could provide a position:
p <- getRatioCoords(ratio=im.r, width=20,
    xleft=20, ybottom=60)
plot_image(img, type='image', add=TRUE,
    xrange=p$x, yrange=p$y)

# Using proportions of plot region:
p <- getRatioCoords(ratio=im.r, height=.5,
    xleft=0, ytop=1, input='prop')
plot_image(img, type='image', add=TRUE,
    xrange=p$x, yrange=p$y)

# Changing the ratio to square:
p <- getRatioCoords(ratio=1, height=.5,
    xright=1, ybottom=0, input='prop')
plot_image(img, type='image', add=TRUE,
    xrange=p$x, yrange=p$y)
# ... and to a long rectangle:
p <- getRatioCoords(ratio=.5, height=1,
    xright=1, ybottom=0, input='prop')
plot_image(img, type='image', add=TRUE,
    xrange=p$x, yrange=p$y,
    replace.colors=list('#B.+'='#FF000033'),
    border='red')
```

`get_palette` *Retrieve the color scheme for contour plots.*

Description

Retrieve the color scheme for contour plots.

Usage

```
get_palette(color, nCol = 50, col = NULL)
```

Arguments

`color`	A string, or vector of strings, indicating a color palette. Includes: 'topo', 'heat', 'bwr', 'cm', 'terrain', 'bpy', 'gray', 'bw', or user defined colors.
`nCol`	The number of colors to use in color schemes.
`col`	Color of contour lines for the contour plots. If NULL (default), a color is determined, depending on the color palette.

Value

Color palette.

See Also

plotsurface, gradientLegend

Other Utility functions: findAbsMin(), find_n_neighbors(), firstLetterCap(), getArrowPos(), getDec(), getRange(), getRatioCoords(), group_sort(), inch2coords(), isColor(), list2str(), move_n_point(), orderBoxplot(), se(), sortGroups()

Examples

```
pal <- get_palette('terrain', nCol=10)
names(pal)
image(matrix(1:10, ncol=10), col=pal$color, axes=FALSE)
# user defined color palette:
pal <- get_palette(c('green', 'orange', 'red'))
image(matrix(1:10, ncol=10), col=pal$color, axes=FALSE)
```

---

gradientLegend *Add a gradient legend to a plot.*

Description

Add a gradient legend to a contour plot (or other plot) to indicate the range of values represented by the color palette.

Usage

```
gradientLegend(
  valRange,
  color = "terrain",
  nCol = 30,
  pos = 0.875,
  side = 4,
  dec = NULL,
  length = 0.25,
  depth = 0.05,
  inside = FALSE,
  coords = FALSE,
  pos.num = NULL,
  n.seg = 1,
  border.col = "black",
  tick.col = NULL,
  fit.margin = TRUE,
  ...
)
```

Arguments

`valRange`	Range of the values that is represented by the color palette. Normally two value-vector. If a larger vector is provided, only the min and max values are being used.
`color`	Name of color palette to use ('topo', 'terrain', 'heat', 'rainbow'). Custom color palettes can also be provided, but then the argument `nCol` is ignored.
`nCol`	Number of colors in the color palette.
`pos`	A number indicating the position on the axis in proportion. Using the arguments `length` and `depth` and `side` the position of the legend is calculated automatically. Alternatively, one could provide a vector with 4 numbers, providing the xleft, ybottom, xright, ytop of a rectangle. These 4 points are indicated in proportions of the x- and y-axis. However, if the argument `coords` is set to TRUE, these positions are taken as values in the Cartesian coordinate system of the plot. Note: `coords` is only considered for 4-number vectors of `pos`.
`side`	Which axis to choose: 1=bottom, 2=left, 3=top, 4=right. Default is 4.
`dec`	Number of decimals for rounding the numbers, set to NULL on default (no rounding).
`length`	Number, indicating the width of the legend as proportion with respect to the axis indicated by `side`. Note: when `pos` is defined by 4 numbers, `length` is ignored.
`depth`	Number, indicating the height of the legend as proportion with respect to the axis perpendicular to `side`. Note: when `pos` is defined by 4 numbers, `depth` is ignored.
`inside`	Logical: whether or not to plot the legend inside or outside the plot area. Note: when `pos` is defined by 4 numbers, `inside` is ignored.
`coords`	Logical: whether or not `pos` (and optionally `n.seg`) is defined as coordinates. When FALSE, the default, `pos` is defined in proportions. Note: when `pos` is defined by 1 number, `inside` is ignored.
`pos.num`	Numeric value, indicating the position of the numbers with respect to the tick marks. 1=bottom, 2=left, 3=top, 4=right.
`n.seg`	Number of ticks and markers on the scale. Defaults to 1. If vector is provided instead of number, all numbers are considered as marker values on the scale provided by `valRange`.
`border.col`	Color of the border (if NA border is omitted).
`tick.col`	Color of the tick marks. Defaults to `border.col` value.
`fit.margin`	Logical: whether the labels of the gradient legend should be forced to fit in the margin or not.
`...`	Other parameters for the marker labels (see `text`).

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# empty plot:
emptyPlot(1,1, main='Test plot', axes=FALSE)
box()
# legend on outside of plotregion:
gradientLegend(valRange=c(-14,14), pos=.5, side=1)
gradientLegend(valRange=c(-14,14), pos=.5, side=2)
gradientLegend(valRange=c(-14,14), pos=.5, side=3)
gradientLegend(valRange=c(-14,14), pos=.5, side=4)

# legend on inside of plotregion:
gradientLegend(valRange=c(-14,14), pos=.5, side=1, inside=TRUE)
gradientLegend(valRange=c(-14,14), pos=.5, side=2, inside=TRUE)
gradientLegend(valRange=c(-14,14), pos=.5, side=3, inside=TRUE)
gradientLegend(valRange=c(-14,14), pos=.5, side=4, inside=TRUE)

# empty plot:
emptyPlot(1,1, main='Test plot', axes=FALSE)
box()
# number of segments:
gradientLegend(valRange=c(-14,14), n.seg=3, pos=.5, side=1)
gradientLegend(valRange=c(-14,14), n.seg=c(-3,5), pos=.5, side=1,
    inside=TRUE)

# This produces a warning, as there is no space for labels here:
## Not run:
    gradientLegend(valRange=c(-14.235,14.2), pos=.5,
        n.seg = c(-7,0), side=4)

## End(Not run)
# different solutions:
# 1. adjust range (make sure also to adjust the range in the plot,
#    for example by changing zlim)
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14,14), n.seg = c(-7,0), side=4)
# 2. reduce number of decimals:
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14.235,14.2), n.seg = c(-7,0), dec=1, side=4)
# 3. change labels to inside plot window:
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14.235,14.2), n.seg = c(-7,0),
    dec=1, side=4, inside=TRUE)
# 4. increase right margin:
oldmar <- par()$mar
par(mar=c(5.1,3.1,4.1,4.1))
```

```
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14.235,14.2), dec=2,
    n.seg = c(-7,0), side=4)
par(mar=oldmar) # return old values
# 5. change label position:
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14.235,14.2), dec=2,
    n.seg = c(-7,0), side=4, pos.num=2)
gradientLegend(valRange=c(-14.235,14.2), dec=2,
    n.seg = c(-7,0), side=4, pos.num=1, pos=.5)
# 6. change legend position and length:
emptyPlot(1,1, main='Test plot')
gradientLegend(valRange=c(-14.235,14.2), dec=2,
    n.seg = c(-7,0), side=3, length=.5, pos=.75)

# change border color (and font color too!)
gradientLegend(valRange=c(-14,14),pos=.75, length=.5,
color=alphaPalette('white', f.seq=seq(0,1, by=.1)),
border.col=alpha('gray'))

# when defining custom points, it is still important to specify side:

gradientLegend(valRange=c(-14,14), pos=c(.5,.25,.7,-.05), coords=TRUE,
    border.col='red', side=1)
gradientLegend(valRange=c(-14,14), pos=c(.5,.25,.7,-.05), coords=TRUE,
    border.col='red', side=2)
```

`group_sort` *Sort split by grouping predictor.*

Description

Function uses `sort.list` to return indices of of a vector, sorted per group.

Usage

```
group_sort(x, group = NULL, decreasing = FALSE)
```

Arguments

`x`	A vector to be sorted.
`group`	A names list that specify the different groups to split the data.
`decreasing`	Logical: whether or not the sort order should be decreasing.

Value

Indices indicating the order of vector x per group.

Author(s)

Jacolien van Rij

See Also

sort.list

Other Utility functions: findAbsMin(), find_n_neighbors(), firstLetterCap(), getArrowPos(), getDec(), getRange(), getRatioCoords(), get_palette(), inch2coords(), isColor(), list2str(), move_n_point(), orderBoxplot(), se(), sortGroups()

Examples

```
# example InsectSprays from R datasets
InsectSprays$Type <- ifelse(InsectSprays$spray %in% c('A','B', 'F'), 1, 2)

ind <- group_sort(InsectSprays$count,
    group=list(Spray=InsectSprays$spray, Type=InsectSprays$Type))
InsectSprays[ind,]
InsectSprays
```

`img` *Image object.*

Description

Map of the Netherlands, png image by Silver Spoon on Wikipedia: https://nl.wikipedia.org/wiki/Bestand:Blank_map_of_the_

Usage

```
img
```

Format

A list with 2 elements:

`image` Matrix of 599 x 564 representing the image.

`col` Vector with colors used in the image.

Author(s)

Jacolien van Rij

inch2coords — *Convert device position (inch) to coordinates in current plot region.*

Description

Convert device position (inch) to coordinates in current plot region.

Usage

```
inch2coords(xpos, ypos = NULL, simplify = FALSE)
```

Arguments

`xpos`	x position in device, inches between position and left side of device. When defined as two-number vector, x- and y-position as measured from bottomleft corner of device.
`ypos`	y position (in inches) from bottom of device.
`simplify`	Logical: whether or not to output a vector instead of a list.

Value

list or 2-number vector

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

isColor — *Check whether color specifications exists.*

Description

Function to check whether all specified colors are actual colors.

Usage

```
isColor(x, return.colors = FALSE)
```

Arguments

`x`	Vector of any of the three kinds of R color specifications, i.e., either a color name (as listed by `palettecolors()`), a hexadecimal string of the form '#rrggbb' or '#rrggbbaa' (see rgb), or a positive integer i meaning `palette()[i]`.
`return.colors`	Logical: logical values (FALSE, default) or returning colors (TRUE)

Value

Logical value (or colors)

Author(s)

Jacolien van Rij

See Also

Other Utility functions: findAbsMin(), find_n_neighbors(), firstLetterCap(), getArrowPos(), getDec(), getRange(), getRatioCoords(), get_palette(), group_sort(), inch2coords(), list2str(), move_n_point(), orderBoxplot(), se(), sortGroups()

Examples

```
# correct color definitions:
isColor(c('#FF0000FF', '#00FF00FF', '#0000FFFF'))
isColor(c('red', 'steelblue', 'green3'))
isColor(c(1,7,28))
# mixtures are possible too:
isColor(c('#FF0000FF', 'red', 1, '#FF0000', rgb(.1,0,0)))

# return colors:
# note that 28 is converted to 4...
isColor(c(1,7,28), return.colors=TRUE)
isColor(c('#FF0000CC', 'red', 1, '#FF0000'), return.colors=TRUE)

# 4 incorrect colors, 1 correct:
test <- c('#FH0000', 3, '#FF00991', 'lavendel', '#AABBCCFFF')
isColor(test)
isColor(test, return.colors=TRUE)
```

---

legend_margin *Add legend with respect to figure instead of plot region. Allows to move legend to margin of plot.*

Description

Add legend with respect to figure instead of plot region. Wrapper around the function legend.

Usage

```
legend_margin(x, legend, adj = NULL, ...)
```

Arguments

`x`	Text string, the location of the legend relative to the figure region. Single keyword from the list 'bottomright', 'bottom', 'bottomleft', 'left', 'topleft', 'top', 'topright', 'right' and 'center'.
`legend`	Vector with text strings to appear in the legend.
`adj`	Numeric vector of length 1 or 2; the string adjustment for legend text.
`...`	Other parameters for specifying the legend (see `legend`).

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
plot(cars$speed, cars$dist, pch=16)
legend_margin('topleft', legend=c('points'), pch=16)
# compare with default legend:
legend('topleft', legend=c('points'), pch=16)
```

`list2str` *Combine list values as string.*

Description

Combine list values as string.

Usage

```
list2str(x, inputlist)
```

Arguments

`x`	A vector with the names or numbers of list elements to be combined.
`inputlist`	A (named) list with information, e.g., graphical parameter settings.

Value

String

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `move_n_point()`, `orderBoxplot()`, `se()`, `sortGroups()`

Examples

```
test <- list(a=c(1,2,3), b='a', c=c(TRUE, FALSE), d='test')
list2str(c('a','c', 'd'), test)
```

`marginDensityPlot` *Plot density of distribution in margins of the plot.*

Description

Plot density of distribution in margins of the plot.

Usage

```
marginDensityPlot(
  x,
  y = NULL,
  side,
  from = NULL,
  scale = 1,
  maxDensityValue = NULL,
  allDensities = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`x`	Density object, or vector with x-values.
`y`	If `x` is not a density object, the vector `y` provides the y-values to plot.
`side`	Number: 1 = bottom, 2 = left, 3 = top, 4 = left
`from`	A number indicating the starting position (bottom) of the density plot. Measured in plot coordinates. Defaults to NULL, which indicate that the border of the plot is taken as the base of the density plot.
`scale`	Scale of the density plot. By default set to 1, which is the size of the margin region.
`maxDensityValue`	Number for scaling the density axis. Default is NULL (automatic scaling fitting the d)
`allDensities`	List with other density objects to determine the plotting scale such that they all fit. Defaults to NULL.

`plot`	Logical: whether to plot the density (default) or not.
`...`	Optional arguments for the lines and fill_area. See `par`.

Author(s)

Jacolien van Rij

See Also

`check_normaldist`

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `plot_error()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# density of a random sample from normal distribution:
val1 <- qnorm(ppoints(500))
val2 <- qt(ppoints(500), df = 2)
dens1 <- density(val1)
dens2 <- density(val2)

# setup plot window:
par(mfrow=c(1,1), cex=1.1)

# increase margin
oldmar <- par()$mar
par(mar=oldmar + c(0,0,0,4))

# plot qqnorm
qqnorm(val2, main='t distribution',
       pch='*', col='steelblue',
       xlim=c(-3,3),
       bty='n')
qqline(val1)
abline(h=0, col=alpha('gray'))
abline(v=0, col=alpha('gray'))

# filled distribution in right margin:
marginDensityPlot(dens2, side=4, allDensities=list(dens1, dens2),
    col='steelblue',lwd=2)
# add lines:
marginDensityPlot(dens2, side=4, allDensities=list(dens1, dens2),
    col='steelblue',density=25, lwd=2)
# compare to normal:
marginDensityPlot(dens1, side=4, allDensities=list(dens1, dens2),
    col=NA, border=1)
# Other sides are also possible:
marginDensityPlot(dens1, side=3, allDensities=list(dens1, dens2),
    col=NA, border=alpha(1), lwd=2)
```

```
marginDensityPlot(dens2, side=3, allDensities=list(dens1, dens2),
    col=NA, border=alpha('steelblue'), lwd=3)
# adjust the starting point with argument 'from' to bottom of plot:
marginDensityPlot(dens1, side=3,
    from=getCoords(0, side=2), lwd=2)
marginDensityPlot(dens2, side=3,
    col='steelblue', from=getCoords(0, side=2), lwd=2,
 maxDensityValue=2*max(dens2$y))

legend(getFigCoords('p')[2], getFigCoords('p')[3],
    yjust=0, legend=c('t distribution', 'Gaussian'),
    fill=c('steelblue', 'black'),
    cex=.75, xpd=TRUE, bty='n')
```

---

move_n_point — *Move a vector n elements forward or backward.*

Description

Move a vector n elements forward or backward.

Usage

```
move_n_point(x, n = 1, na_value = NA)
```

Arguments

x	A vector.
n	Number indicating how many steps the vector should shift forward (N > 0) or backward (n < 0).
na_value	The value to replace the empty cells with (e.g., the first or last points). Defaults to NA.

Value

A vector with the same length of x, all moved n steps.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: findAbsMin(), find_n_neighbors(), firstLetterCap(), getArrowPos(), getDec(), getRange(), getRatioCoords(), get_palette(), group_sort(), inch2coords(), isColor(), list2str(), orderBoxplot(), se(), sortGroups()

Examples

```
(x <- -10:30)
prev <- move_n_point(x)
change <- x - prev
post5 <- move_n_point(x, n=-5)

emptyPlot(length(x), range(x))
lines(x)
lines(prev, col='red')
lines(post5, col='blue')
```

`orderBoxplot` *Order boxplot stats following a given ordering.*

Description

Order boxplot stats following a given ordering.

Usage

```
orderBoxplot(stats, idx)
```

Arguments

`stats`	List with information produced by a box-and-whisker plot.
`idx`	Order of group levels.

Value

The ordered stats.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `se()`, `sortGroups()`

Examples

```
head(ToothGrowth)
# sort on basis of mean length:
bp <- boxplot(len ~ dose:supp, data = ToothGrowth, plot=FALSE)
idx <- sortGroups(len ~ dose:supp, data = ToothGrowth)
bp2 <- orderBoxplot(bp, idx)
# compare:
bp$names
bp2$names
```

`plotfunctions` *Package plotfunctions: Various Functions to Facilitate Visualization of Data and Analysis*

Description

This package provides a set of simple tools for building plots incrementally, starting with an empty plot region, and adding bars, data points, regression lines, error bars, gradient legends, density distributions in the margins, and even pictures. The package builds further on R graphics by simply combining functions and settings in order to reduce the amount of code to produce for the user. As a result, the package does not use formula input or special syntax, but can be used in combination with default R plot functions.

Details

Note: Most of the functions were part of the package itsadug, which is now split in two packages: 1. the package `itsadug`, which contains the core functions for visualizing and evaluating nonlinear regression models, and 2. the package `plotfunctions`, which contains more general plot functions.

See `vignette(package='plotfunctions', 'plotfunctions')` for an overview with examples.

Basic functions

- `emptyPlot` generates an empty plot.
- `plot_error` adds line with (shaded) confidence interval.
- `add_bars` adds bars to a (bar)plot.
- `errorBars` adds confidence intervals to points or bars.

Specialized plots

- `color_contour` and `plotsurface` are wrappers around `image` and `contour` for making easily colored surface plots for interactions with two (or more) continuous predictors.
- `plot_image` can be used to add a picture to a plot, or to make a picture the background of a plot.
- `marginDensityPlot` adds distributions in the margins of the plot.
- `check_normaldist` overlays density of data to normal distribution. Might help with interpretation of QQ-plots that are generally used to test for normality.

Other useful features

- `alpha` and `alphaPalette` are simple function to make colors and palettes transparent.
- `legend_margin` adds a legend in the margins of a plot.
- `gradientLegend` adds a color legend to a plot.
- `drawDevArrows` for drawing arrows or lines between different panels.
- `getFigCoords` retrieve the cartesian coordinates relative to the plot axes for given proportions of the plot region or given proportions of the figure. #'
- A list of all available functions is provided in `help(package='itsadug')`.

Author(s)

Jacolien van Rij

Maintainer: Jacolien van Rij (<`vanrij.jacolien@gmail.com`>)

University of Groningen, The Netherlands & University of Tuebingen, Germany

---

`plotsurface` *Creates a colored surface plot from data frame input.*

Description

This function is a wrapper around `image` and `contour`. See `vignette('plotfunctions')` for an example of how you could use `image` and `contour`.

Usage

```
plotsurface(
  data,
  view,
  predictor = NULL,
  valCI = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  col = NULL,
  color = terrain.colors(50),
  ci.col = c("green", "red"),
  nCol = 50,
  add.color.legend = TRUE,
  dec = NULL,
  fit.margin = TRUE,
  ...
)
```

Arguments

`data`	Data frame or list with plot data. A data frame needs to have a column with x values, a column with y values and a column with z values. A list contains a vector with unique x values, a vector with unique y values, and a matrix with z-values. The output of the function fvisgam is an example of a suitable list.
`view`	A vector with the names or numbers of the columns to plot on the x axis and y axis respectively.
`predictor`	Optional: the name of the column in the data frame `data` that provides the z-values. If data contains more than one column besides the x- and y-values, the `predictor` should be provided.
`valCI`	Optional: the name of the column in the data frame `data` that provides the CI-values. If not NULL, CI contour lines will be plotted.
`main`	Text string, an overall title for the plot.
`xlab`	Label for x axis. Default is name of first `view` variable.
`ylab`	Label for y axis. Default is name of second `view` variable.
`xlim`	x-limits for the plot.
`ylim`	y-limits for the plot.
`zlim`	z-limits for the plot.
`col`	Color for the contour lines and labels.
`color`	The color scheme to use for plots. One of 'topo', 'heat', 'cm', 'terrain', 'gray', 'bwr' (blue-white-red) or 'bw'. Or a list of colors such as that generated by `rainbow`, `heat.colors` `colors`, `topo.colors`, `terrain.colors` or similar functions. Alternatively a vector with some colors can be provided for a custom color palette.
`ci.col`	Two-value vector with colors for the lower CI contour lines and for the upper CI contour lines.
`nCol`	The number of colors to use in color schemes.
`add.color.legend`	Logical: whether or not to add a color legend. Default is TRUE. If FALSE (omitted), one could use the function `gradientLegend` to add a legend manually at any position.
`dec`	Numeric: number of decimals for rounding the color legend. When NULL (default), no rounding. If -1 (default), automatically determined. Note: if value = -1 (default), rounding will be applied also when `zlim` is provided.
`fit.margin`	Logical: whether the labels of the gradient legend should be forced to fit in the margin or not.
`...`	Optional parameters for `image` and `contour`.

Author(s)

Jacolien van Rij

See Also

image, contour, color_contour

Other Functions for plotting: addInterval(), add_bars(), add_n_points(), alphaPalette(), alpha(), check_normaldist(), color_contour(), dotplot_error(), drawDevArrows(), emptyPlot(), errorBars(), fill_area(), getCoords(), getFigCoords(), getProps(), gradientLegend(), legend_margin(), marginDensityPlot(), plot_error(), plot_image(), sortBoxplot()

Examples

```
# From the package graphics, see help(image):
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))
image(x, y, volcano, col = terrain.colors(100), axes = FALSE)
contour(x, y, volcano, levels = seq(90, 200, by = 5),
        add = TRUE, col = 'peru')
axis(1, at = seq(100, 800, by = 100))
axis(2, at = seq(100, 600, by = 100))
box()
title(main = 'Maunga Whau Volcano', font.main = 4)

# now with plot surface:
# first convert to data frame
tmp <- data.frame(value = as.vector(volcano),
    x = 10*rep(1:nrow(volcano), ncol(volcano)),
    y = 10*rep(1:ncol(volcano), each=nrow(volcano)))
plotsurface(tmp, view=c('x', 'y'), predictor='value',
    main='Maunga Whau Volcano')

# or with gray scale colors:
plotsurface(tmp, view=c('x', 'y'), predictor='value',
    main='Maunga Whau Volcano', color='gray')

# change color range:
plotsurface(tmp, view=c('x', 'y'), predictor='value',
    main='Maunga Whau Volcano', zlim=c(0,200))

#' remove color and color legend:
plotsurface(tmp, view=c('x', 'y'), predictor='value',
    main='Maunga Whau Volcano',
    color=NULL, col=1, add.color.legend=FALSE)
```

plot_error *Utility function*

Description

Plot line with confidence intervals.

Usage

```
plot_error(
  x,
  fit,
  se.fit,
  se.fit2 = NULL,
  shade = FALSE,
  f = 1,
  col = "black",
  ci.lty = NULL,
  ci.lwd = NULL,
  border = FALSE,
  alpha = 0.25,
  ...
)
```

Arguments

`x`	Vector with values on x-axis.
`fit`	Vector with values on y-axis.
`se.fit`	Vector with standard error; or when `se.fit2` is provided, `se.fit` specifies upper values confidence interval.
`se.fit2`	Optional: lower values confidence interval.
`shade`	Logical: whether or not to produce shaded regions as confidence bands.
`f`	Factor for converting standard error in confidence intervals. Defaults to 1. Use 1.96 for 95% CI, and 2.58 for 99% CI.
`col`	Color for lines and confindence bands.
`ci.lty`	Line type to be used for the error lines, see `par`.
`ci.lwd`	Line type to be used for the error lines, see `par`.
`border`	The color to draw the border for the shaded confidence interval. The default, FALSE, omits borders.
`alpha`	Transparency of shaded area. Number between 0 (completely transparent) and 1 (not transparent).
`...`	Optional arguments for the lines and shaded area.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_image()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# generate some data:
x <- -10:20
y <- 0.3*(x - 3)^2 + rnorm(length(x))
s <- 0.2*abs(100-y + rnorm(length(x)))

# Plot line and standard deviation:
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s)
# Change layout:
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, shade=TRUE, lty=3, lwd=3)

# Use of se.fit2 for asymmetrical error bars:
cu <- y + .65*s
cl <- y - s
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, shade=TRUE)
plot_error(x, y, se.fit=cu, se.fit2=cl, col='red', shade=TRUE)

# Some layout options:
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, lty=3, lwd=1, ci.lty=1, ci.lwd=3)
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, shade=TRUE, lty=3, lwd=3)
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, shade=TRUE, lty=1, lwd=3, ci.lwd=3, border='red')
emptyPlot(range(x), range(y), h0=0)
plot_error(x, y, s, shade=TRUE, lty=1, lwd=3, density=10, ci.lwd=3)
```

---

plot_image *Add images to plots.*

Description

Add images to plots.

Usage

```
plot_image(
  img,
  type = "image",
  col = NULL,
  show.axes = FALSE,
  xrange = c(0, 1),
  yrange = c(0, 1),
  keep.ratio = FALSE,
  adj = 0,
  fill.plotregion = FALSE,
  replace.colors = NULL,
  add = FALSE,
  interpolate = TRUE,
  ...
)
```

Arguments

`img`	Matrix or image object (list with 'image', a matrix, and 'col', a vector with color values), or a string indicating the filename of an image to read.
`type`	String, 'image' (default), 'png', 'jpeg', 'gif'
`col`	Vector with colors.
`show.axes`	Logical: whether or not to plot the axes.
`xrange`	Two-value vector providing the xleft and xright coordinate values of the picture. Default set to c(0,1).
`yrange`	Two-value vector providing the ybottom and ytop coordinate values of the picture. Default set to c(0,1).
`keep.ratio`	Logical: whether or not to keep the original picture ratio.
`adj`	Numeric value indicating the position of the shortest picture side with respect to `xrange` or `yrange`. Only applies when `keep.ratio=TRUE`. See examples.
`fill.plotregion`	Logical: whether or not to fill the complete plot region. Defaults to FALSE.
`replace.colors`	Named list for replacing colors. The names are the colors (in hexadecimal values), or regular expressions matching colors. The values are the replacements.
`add`	Logical: whether or not to add the plot to the current plot.
`interpolate`	Logical: a logical vector (or scalar) indicating whether to apply linear interpolation to the image when drawing.
`...`	Other arguments for plotting, see `par`.

Value

Optionally returns

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plotsurface()`, `sortBoxplot()`

Examples

```
# see Volcano example at help(image)
# create image object:
myimg <- list(image=volcano-min(volcano), col=terrain.colors(max(volcano)-min(volcano)))
# create emoty plot window:
emptyPlot(1,1, main='Volcano images')
# add image topleft corner:
plot_image(img=myimg, xrange=c(0,.25), yrange=c(.75,1), add=TRUE)
# add transparent image as overlay:
myimg$col <- alpha(myimg$col, f=.25)
plot_image(img=myimg, add=TRUE, fill.plotregion=TRUE, bty='n')
# add image:
myimg$col <- topo.colors(max(myimg$image))
plot_image(img=myimg, xrange=c(0.125,.375), yrange=c(.5,.875), add=TRUE)
# add some points and lines:
points(runif(10,0,1), runif(10,0,1), type='o')

# keep ratio:
emptyPlot(1,1, main='Volcano images')
# I would like to add an image in the following field:
rect(xleft=0, xright=.5, ybottom=0, ytop=.3, col='gray', border=NA)
# add image with keep.ratio=true
plot_image(img=myimg, xrange=c(0,.5), yrange=c(0,.3),
    add=TRUE, keep.ratio=TRUE, border=NA)
# as y-side is longest, this side will be fitted in
# the rectangle and the x position adjusted with adj:
plot_image(img=myimg, xrange=c(0,.5), yrange=c(0,.3),
    add=TRUE, keep.ratio=TRUE, border=2, adj=0.5)
plot_image(img=myimg, xrange=c(0,.5), yrange=c(0,.3),
    add=TRUE, keep.ratio=TRUE, border=3, adj=1)

# keep.ratio and border:
plot_image(img=myimg, xrange=c(0,1), yrange=c(0,1),
    keep.ratio=TRUE, adj=0.5)
plot_image(img=myimg, xrange=c(0,.5), yrange=c(0,1),
    keep.ratio=TRUE, adj=0.5)
emptyPlot(1,1, axes=FALSE)
plot_image(img=myimg, xrange=c(0,1), yrange=c(0,1),
    add=TRUE, keep.ratio=TRUE, adj=0.5)
```

`plot_signifArea` *Creates a colored surface plot from data frame input.*

Description

This function uses `rasterImage` to indicate which points in the surface are not significantly different from zero. Note that the shape of these non-significant regions depends on the number of data points (often specified with `n.grid`).

Usage

```
plot_signifArea(data, view, predictor = NULL, valCI, col = 1, alpha = 0.5, ...)
```

Arguments

`data`	Data frame with plot data. A data frame needs to have a column with x values, a column with y values (specified in `view`), a column with z values (`predictor`), and one or two columns with CI values (`valCI`).
`view`	A vector of length 2 with the names or numbers of the columns to plot on the x axis and y axis respectively.
`predictor`	The name of the column in the data frame `data` that provides the z-values. If data contains more than one column besides the x- and y-values, the `predictor` should be provided.
`valCI`	The name of the column in the data frame `data` that provides the CI-values. Alternatively, two column names can be provided for the lower and upper CI respectively.
`col`	Color for the nonsignificant areas.
`alpha`	Level of transparency, number between 0 (transparent) and 1 (no transparency)
`...`	Optional parameters for `rasterImage`

Author(s)

Jacolien van Rij

See Also

`rasterImage`

Examples

```
# From the package graphics, see help(image):
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))
tmp <- data.frame(value = (as.vector(volcano) - 120),
    x = 10*rep(1:nrow(volcano), ncol(volcano)),
    y = 10*rep(1:ncol(volcano), each=nrow(volcano)),
    CI = rep(20, nrow(volcano)*ncol(volcano)))
plotsurface(tmp, view=c('x', 'y'), predictor='value', main='Maunga Whau Volcano')
plot_signifArea(tmp, view=c('x', 'y'), predictor='value', valCI='CI')

# change color:
plotsurface(tmp, view=c('x', 'y'), predictor='value', main='Maunga Whau Volcano')
plot_signifArea(tmp, view=c('x', 'y'), predictor='value', valCI='CI',
    col='red')
# or completely remove 'nonsignificant' area:
plot_signifArea(tmp, view=c('x', 'y'), predictor='value', valCI='CI',
    col='white', alpha=1)
```

se — *Calculate standard error of the mean.*

Description

Calculate standard error of the mean.

Usage

```
se(x, na.rm = FALSE)
```

Arguments

`x`	A vector.
`na.rm`	Logical: whether or not to remove NA values (default set to FALSE - including NAs).

Value

Standard Error of the mean.

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `sortGroups()`

Examples

```
# load example data:
data(chickwts)
str(chickwts)

# first calculate means per feeding type:
avg <- with(chickwts, tapply(weight, list(feed), mean))
par(cex=1.25)
b <- barplot(avg, beside=TRUE, names.arg=FALSE, ylim=c(0,450))
text(b, rep(0, length(b)), labels=names(avg), srt=90, adj=-.25)

# calculate mean collapsing over feeding types:
abline(h=mean(avg), lwd=1.5, col='red1')
# add SE reflecting variation between feeding types:
abline(h=mean(avg)+c(-1,1)*se(avg), lty=2, col='red1')
text(getCoords(.5), mean(avg)+se(avg),
    labels=expression('mean' %+-% '1SE'), pos=3, col='red1')

# Note that SE makes more sense for experiments with
# different groups or participants.
```

`sortBoxplot`	*Produce box-and-whisker plot(s) ordered by function such as mean or median.*

Description

Produce box-and-whisker plot(s) ordered by function such as mean or median. Wrapper around `boxplot`.

Usage

```
sortBoxplot(
  formula,
  data = NULL,
  decreasing = TRUE,
  FUN = "median",
  idx = NULL,
  col = "gray",
  ...
)
```

Arguments

`formula`	a formula, such as 'y ~ Condition', where 'y' is a numeric vector of data values to be split into groups according to the grouping variable 'Condition' (usually a factor).
`data`	a data.frame from which the variables in 'formula' should be taken.
`decreasing`	Logical: Indicating whether the sort should be increasing or decreasing.
`FUN`	a function to compute the summary statistics which can be applied to all data subsets.
`idx`	Numeric vector providing the ordering of groups instead of specifying a function in `FUN`. Only is used when `FUN` is set to NULL.
`col`	Fill color of the boxplots. Alias for `boxfill`.
`...`	Other parameters to adjust the layout of the boxplots. See `bxp`.

Value

The ordered stats.

Author(s)

Jacolien van Rij

See Also

Other Functions for plotting: `addInterval()`, `add_bars()`, `add_n_points()`, `alphaPalette()`, `alpha()`, `check_normaldist()`, `color_contour()`, `dotplot_error()`, `drawDevArrows()`, `emptyPlot()`, `errorBars()`, `fill_area()`, `getCoords()`, `getFigCoords()`, `getProps()`, `gradientLegend()`, `legend_margin()`, `marginDensityPlot()`, `plot_error()`, `plot_image()`, `plotsurface()`

Examples

```
head(ToothGrowth)
# sort on basis of mean length:
sortBoxplot(len ~ dose:supp, data = ToothGrowth)
# sort on basis of median length:
sortBoxplot(len ~ dose:supp, data = ToothGrowth, decreasing=FALSE)
# on the basis of variation (sd):
sortBoxplot(len ~ dose:supp, data = ToothGrowth, FUN='sd', col=alpha(2))
```

`sortGroups` *Sort groups based on a function such as mean value or deviation.*

Description

Sort groups based on a function such as mean value or deviation.

Usage

```
sortGroups(formula, FUN = "mean", decreasing = FALSE, ...)
```

Arguments

`formula`	Formula for splitting the data
`FUN`	Function to apply to each group
`decreasing`	Logical: sort groups on decreasing values or not (default is FALSE, sorting on increasing values).
`...`	Additional arguments for the function `aggregate`.

Value

The order of levels.

Author(s)

Jacolien van Rij

See Also

Other Utility functions: `findAbsMin()`, `find_n_neighbors()`, `firstLetterCap()`, `getArrowPos()`, `getDec()`, `getRange()`, `getRatioCoords()`, `get_palette()`, `group_sort()`, `inch2coords()`, `isColor()`, `list2str()`, `move_n_point()`, `orderBoxplot()`, `se()`

Examples

```
head(ToothGrowth)
# sort on basis of mean length:
sortGroups(len ~ dose:supp, data = ToothGrowth)
labels = levels(interaction(ToothGrowth$dose, ToothGrowth$supp))
labels[sortGroups(len ~ dose:supp, data = ToothGrowth)]
```

Index