Package ‘BoutrosLab.plotting.general’

October 4, 2024

Version 7.1.2

Type Package

Title Functions to Create Publication-Quality Plots

Date 2024-10-02

Maintainer Paul Boutros <`PBoutros@mednet.ucla.edu`>

Depends R (>= 3.5.0), lattice (>= 0.20-35), latticeExtra (>= 0.6-27), cluster (>= 2.0.0), hexbin (>= 1.27.0), grid

Imports gridExtra, tools, methods, gtable, e1071, MASS(>= 7.3-29)

Suggests Cairo (>= 1.5-1), knitr, testthat (>= 3.0.0)

Description

Contains several plotting functions such as barplots, scatterplots, heatmaps, as well as functions to combine plots and assist in the creation of these plots. These functions will give users great ease of use and customization options in broad use for biomedical applications, as well as general purpose plotting. Each of the functions also provides valid default settings to make plotting data more efficient and producing high quality plots with standard colour schemes simpler. All functions within this package are capable of producing plots that are of the quality to be presented in scientific publications and journals. P'ng et al.; BPG: Seamless, automated and interactive visualization of scientific data; BMC Bioinformatics 2019 <doi:10.1186/s12859-019-2610-2>.

License GPL-2

URL `https://github.com/uclahs-cds/package-BoutrosLab-plotting-general`

BugReports

`https://github.com/uclahs-cds/package-BoutrosLab-plotting-general/issues`

LazyLoad yes

LazyData yes

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Paul Boutros [aut, cre],
Christine P'ng [ctb],
Jeff Green [ctb],
Stephenie Prokopec [ctb],
Ontario Institute for Cancer Research [cph],
The R Core Team [cph],
The R Foundation [cph],
Robert Gentleman [ctb],
Ross Ihaka [ctb],
Caden Bugh [ctb],
Dan Knight [ctb],
Stefan Eng [ctb],
Mohammed Faizal Eeman Mootor [ctb],
Rachel Dang [ctb],
John Sahrmann [ctb]

Repository CRAN

Date/Publication 2024-10-04 04:50:08 UTC

Contents

`auto.axis` *Create ideal labels and values for a given numeric vector (detects log scales)*

Description

Takes a numeric vector and several parameters and outputs an object with values and labels ideal for given data

Usage

```
auto.axis(
    x,
    pretty = TRUE,
    log.scaled = NA,
    log.zero = 0.1,
    max.factor = 1,
    min.factor = 1,
    include.origin = TRUE,
    num.labels = 5,
    max.min.log10.diff = 2
)
```

Arguments

`x`	Numeric vector to be scaled
`pretty`	Parameter flag for if output should be in pretty format
`log.scaled`	parameter set to determine if scaling is logarithmic or not
`log.zero`	log 0 starting point
`max.factor`	maximum factor for y variable
`min.factor`	minimum factor for y variable
`include.origin`	flag to include the origin value or not
`num.labels`	number of labels to output
`max.min.log10.diff`	the max and min diffrence for dataset to be determined logarithmic

Author(s)

Takafumi N. Yamaguchi

See Also

`stripplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(223);
simple.data <- data.frame(
    x = sample(1:15, 10),
    y = LETTERS[1:10]
    );

auto.axis(simple.data$x)

data2 <- c(1,10,100,1000)

auto.axis(data2)
```

`CNA` *Copy number aberration (CNA) data from colon cancer patients*

Description

CNA calls from 30 genes across 58 colon cancer patients. Additional data on the patient samples is found in the `patient` dataset. The same patient samples are described in the `microarray` and `SNV` datasets.

Usage

```
CNA
```

Format

A data frame with 58 columns and 30 rows. The columns indicate the patient sample, and the rows indicate the gene. The contents of the data frame are encoded such that 0 indicates no CNA, -1 indicates a CNA loss, and 1 indicates a CNA gain.

Author(s)

Christine P'ng

Examples

```
data(CNA);
create.dotmap(
    # filename = tempfile(pattern = 'Using_CNA_dataset', fileext = '.tiff'),
    x = CNA[1:15, 1:15],
    main = 'CNA data',
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.rot = 90,
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 50
    );
```

`colour.gradient` *Creates a colour gradient*

Description

Creates a sequential palette of colours.

Usage

```
colour.gradient(
colour,
length
);
```

Arguments

`colour`	A single colour to be used as the center value of the sequence
`length`	The number of colours to include in the palette

Author(s)

Ren Sun & Christine P'ng

Examples

```
display.colours(colour.gradient('dodgerblue2', 6));

display.colours(colour.gradient(default.colours(1), 3));
```

`covariates.grob` *Create one or more covariate bars*

Description

Takes a list of covariate bar annotates and creates a grid graphical object for them

Usage

```
covariates.grob(
covariates,
ord,
side = 'right',
size = 1,
grid.row = NULL,
grid.col = NULL,
grid.border = NULL,
row.lines = NULL,
col.lines = NULL,
reorder.grid.index = FALSE,
        x = 0.5,
        y = 0.5
);
```

Arguments

`covariates`	Any covariate annotate to add to the plot, as a fully formed list.
`ord`	A vector of integer indices indicating the order of the items in the covariate bars.
`side`	Intended position of the covariate bar when added as a legend. Allowed positions are “right” and “top”.
`size`	The size of each covariate bar in units of “lines”.
`grid.row`	A list of parameters to be passed to `gpar` specifying the behaviour of row lines in the covariate bars. See `Notes` for details.
`grid.col`	A list of parameters to be passed to `gpar` specifying the behaviour of column lines in the covariate bars.
`grid.border`	A list of parameters to be passed to `gpar` specifying the behaviour of the border around the covariate bars.
`row.lines`	Vector of row indices where grid lines should be drawn. If NULL (default), all row lines are drawn. Ignored if `grid.row` is not specified.

`col.lines`	Vector of column indices where grid lines should be drawn. If NULL (default), all column lines are drawn. Ignored if `grid.col` is not specified.
`reorder.grid.index`	Boolean specifying whether grid line indices should be re-ordered according to the `ord` argument. Defaults to FALSE.
`x`	x coordinate in npc coordinate system
`y`	y coordinate in npc coordinate system

Value

A grid graphical object (grob) representing the covariate bar(s)

Notes

This code is an adaptation of the `dendrogramGrob` function in the `latticeExtra` package. It uses functions of the `grid` package.

By default, the covariate bar grid is drawn via borders around individual rectangles using the parameters specified in the `covariates` argument (col, lwd, etc.). If `grid.row`, `grid.col`, or `grid.border` are specified by the user, additional grid lines are drawn over any existing ones using the parameters in these lists.

Author(s)

Lauren Chong

See Also

`gpar`

Examples

```
# The 'cairo' graphics is preferred but on M1 Macs this is not available
bitmap.type = getOption('bitmapType')
if (capabilities('cairo')) {
bitmap.type <- 'cairo';
}

# create temp data
set.seed(1234567890);

x <- outer(-5:5, -5:5, '*') + matrix(nrow = 11, ncol = 11, data = runif(11 * 11));
colnames(x) <- paste('col', 1:11, sep = '-');
rownames(x) <- paste('row', 1:11, sep = '-');

# set covariates
covariate.colours1 <- x[,1]
covariate.colours1[covariate.colours1 >= 0] <- default.colours(3)[1];
covariate.colours1[covariate.colours1 != default.colours(3)[1]] <- default.colours(3)[2];

covariate.colours2 <- x[,1]
```

```
covariate.colours2[covariate.colours2 >= 0] <- default.colours(3)[2];
covariate.colours2[covariate.colours2 != default.colours(3)[2]] <- default.colours(3)[3];

# create an object to draw the covariates from
covariates1 <- list(
    rect = list(
        col = 'black',
        fill = covariate.colours1,
        lwd = 1.5
        ),
    rect = list(
        col = 'black',
        fill = covariate.colours2,
        lwd = 1.5
        )
    );

# create a covariates grob using a simple incremental ordering and default behaviour
covariates.grob1 <- covariates.grob(
    covariates = covariates1,
    ord = c(1:ncol(x)),
    side = 'right'
    );

# create a dendrogram for x
cov.dendrogram <- BoutrosLab.plotting.general::create.dendrogram(
    x = x,
    clustering.method = 'average'
    );

covariates2 <-list(
    rect = list(
        col = 'black',
        fill = covariate.colours2,
        lwd = 1.5
        )
    );

# create a covariates grob using the dendrogram ordering and double the default size
covariates.grob2 <- covariates.grob(
    covariates = covariates2,
    ord = order.dendrogram(cov.dendrogram),
    side = 'top',
    size = 2
    );

# add a border of a different colour
covariates.grob3 <- covariates.grob(
    covariates = covariates1,
    ord = c(1:ncol(x)),
    side = 'right',
    grid.border = list(col = 'red', lwd = 1.5)
    );
```

```
# create covariates with transparent rectangle borders
covariates3 <- list(
    rect = list(
        col = 'transparent',
        fill = covariate.colours1,
        lwd = 1.5
        ),
    rect = list(
        col = 'transparent',
        fill = covariate.colours2,
        lwd = 1.5
        )
    );

# add column grid lines and a border with default gpar settings
covariates.grob4 <- covariates.grob(
    covariates = covariates3,
    ord = c(1:nrow(x)),
    side = 'top',
    grid.col = list(col = 'black', lty = 3),
    grid.border = list()
    );

# draw a subset of row/column lines
covariates.grob5 <- covariates.grob(
    covariates = covariates3,
    ord = order.dendrogram(cov.dendrogram),
    side = 'right',
    grid.row = list(lineend = 'butt', lwd = 2),
    row.lines = 6,
    reorder.grid.index = FALSE, # note: this is already set by default
    grid.col = list(lty = 2),
    col.lines = c(0,1)
    );
```

`create.barplot` *Make a barplot*

Description

Takes a data.frame and creates a barplot

Usage

```
create.barplot(
formula,
data,
groups = NULL,
stack = FALSE,
```

```
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
abline.h = NULL,
abline.v = NULL,
abline.lty = 1,
abline.lwd = NULL,
abline.col = 'black',
axes.lwd = 1,
add.grid = FALSE,
xgrid.at = xat,
ygrid.at = yat,
grid.lwd = 5,
grid.col = NULL,
xaxis.lab = TRUE,
yaxis.lab = TRUE,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.tck = 1,
yaxis.tck = 1,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
layout = NULL,
as.table = FALSE,
x.spacing = 0,
```

```
y.spacing = 0,
x.relation = 'same',
y.relation = 'same',
top.padding = 0.5,
bottom.padding = 1,
right.padding = 1,
left.padding = 1,
key.bottom = 0.1,
ylab.axis.padding = 0.5,
xlab.axis.padding = 0.5,
col = 'black',
border.col = 'black',
border.lwd = 1,
plot.horizontal = FALSE,
background.col = 'transparent',
origin = 0,
reference = TRUE,
box.ratio = 2,
sample.order = 'none',
group.labels = FALSE,
key = list(text = list(lab = c(''))),
legend = NULL,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
strip.col = 'white',
strip.cex = 1,
y.error.up = NULL,
y.error.down = y.error.up,
y.error.bar.col = 'black',
error.whisker.width = width/(nrow(data)*4),
error.bar.lwd = 1,
error.whisker.angle = 90,
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'grey85',
alpha.rectangle = 1,
line.func = NULL,
line.from = 0,
line.to = 0,
line.col = 'transparent',
```

```
line.infront = TRUE,
text.above.bars = list(labels = NULL,
padding = NULL,
bar.locations = NULL,
rotation = 0
),
raster = NULL,
raster.vert = TRUE,
raster.just = 'center',
raster.width.dim = unit(2/37, 'npc'),
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
inside.legend.auto = FALSE,
disable.factor.sorting = FALSE
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame. Transforming data within formula is not compatible with automatic scaling with ‘xat‘ or ‘yat‘
`data`	The data-frame to plot
`groups`	Optional grouping variable. Expression or variable.
`stack`	Logical, relevant when groups is non-null. If FALSE (the default), bars for different values of the grouping variable are drawn side by side, otherwise they are stacked
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 3
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to black

`ylab.col`	Colour of the y-axis label, defaults to black
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`abline.h`	Specify the superimposed horizontal line(s)
`abline.v`	Specify the superimposed vertical line(s)
`abline.lty`	Specify the superimposed line type
`abline.lwd`	Specify the superimposed line width
`abline.col`	Specify the superimposed line colour (defaults to black)
`axes.lwd`	Specify line width of the axes; set to 0 to turn off axes
`add.grid`	Specify whether to draw grid or not (defaults to FALSE)
`xgrid.at`	Specify where to draw x-axis grid lines (defaults to xat)
`ygrid.at`	Specify where to draw y-axis grid lines (defaults to yat)
`grid.lwd`	Specify width of grid line (defaults to 5)
`grid.col`	Specify colour of grid line. Currently only supports one colour. Defaults to NULL, which uses the colour of the reference line.
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with xat will overwrite user input. Set to NULL to remove x-axis labels.
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with yat will overwrite user input. Set to NULL to remove y-axis labels.
`xaxis.col`	Colour of the x-axis tick labels, defaults to black
`yaxis.col`	Colour of the y-axis tick labels, defaults to black
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.cex`	Size of x-axis tick labels, defaults to 1.2
`yaxis.cex`	Size of y-axis tick labels, defaults to 1.5
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xlimits`	Two-element vector giving the x-axis limits. Useful when plot.horizontal = TRUE
`ylimits`	Two-element vector giving the y-axis limits

`xat`	Accepts a vector listing where x-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes x-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘. Useful when plot.horizontal = TRUE
`yat`	Accepts a vector listing where y-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes y-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
.	
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`top.padding`	A number specifying the distance to the top margin, defaults to 0.5
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 2
`right.padding`	A number specifying the distance to the right margin, defaults to 1
`left.padding`	A number specifying the distance to the left margin, defaults to 1
`key.bottom`	A number specifying how much space should be left for the key at the bottom, defaults to 0.1
`ylab.axis.padding`	A number specifying the distance of y-axis label to the y-axis, defaults to 0
,	
`xlab.axis.padding`	A number specifying the distance of x-axis label to the x-axis, defaults to 0.5. Named differently than ylab.axis.padding because these are lattice’s internal names for these values
`col`	Filling colour of bars, defaults to black, does a grey-scale spectrum if !is.null(groups)
`border.col`	Specify border colour (defaults to black)
`border.lwd`	Specify border width (defaults to 1)

`plot.horizontal`	Plot the bars horizontally. Note if `disable.factor.sorting = TRUE`, then the top row of `data` is the bottom row of the plot, i.e. bars are filled in from the bottom to the top of the plot. To make the barplot rows match the input data rows, make sure the y-axis variable is a factor, and do `data = data[nrow(data):1,]`
`background.col`	Plot background colour, defaults to transparent
`origin`	The origin of the plot, generally 0
`reference`	Should the reference line be printed at the origin
`box.ratio`	Specifies the width of each bar, defaults to 2
`sample.order`	Should the bars be reordered, accepts values “increasing”, “decreasing” or a vector of sample names. Labels will also be reordered
`group.labels`	Should the labels be grouped to the same amount of bars per column
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`strip.col`	Strip background colour, defaults to white
`strip.cex`	Strip title character expansion
`y.error.up`	A vector specifying the length of the error bar going up from each point. If set to NULL (the default), error bars will not be drawn
`y.error.down`	A vector specifying the length of the error bar going down from each point. By default, it is set to y.error.up
`y.error.bar.col`	A string or vector of strings specifying the colour of the error bars. Defaults to black
`error.whisker.width`	A number specifying the width of the error bars. Defaults to a rough approximation based on the size of the data
`error.bar.lwd`	The line width of the error bars. Defaults to 1
`error.whisker.angle`	The angle of the error bar whiskers, defaults to 90. Can be changed to produce arrow-like bars
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn

`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle
`line.func`	Function for the line that should be drawn on top of plot
`line.from`	The starting point of the line on the plot
`line.to`	The ending point of the line on the plot
`line.col`	Colour of the line on the plot
`line.infront`	Should the line appear in front of the plot or not
`text.above.bars`	Should some form of text appear above the bars; input as a list. `bar.locations` is the x-axis when vertical and y-axis when horizontal. See `lattice::ltext` arguments for all possible values that can be passed in. (`col`, `alpha`, `cex`, etc, can all be passed in as a single value or vector of same length as `text.above.bars$labels`)
`raster`	The image to raster over each bar - see Raster Images in R Graphics by Paul Murrell for full details
`raster.vert`	A logical indicating whether the raster is applied vertically or horizontally
`raster.just`	A word giving the justification of the raster, can be set to "left", "right", "centre", "center", "bottom", or "top"
`raster.width.dim`	A unit object giving the width of the raster bar
`height`	Figure height, defaults to 6 in
`width`	Figure width, defaults to 6 in
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Description of image/plot; default NULL
`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function
`disable.factor.sorting`	Disable barplot auto sorting factors alphabetically/numerically

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Mehrdad Shamsi

See Also

`barchart`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = sample(1:15, 5),
    y = LETTERS[1:5]
    );

# Simple example
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Simple', fileext = '.tiff'),
    formula = x ~ y,
    data = simple.data,
    yat = seq(0,16,2),
    resolution = 30
    );

# set up the data
total.counts <- apply(SNV[1:15], 2, function(x){ mutation.count <- (30 - sum(is.na(x)))});
count.nonsyn <- function(x){
    mutation.count <- length(which(x == 1));
    }
nonsynonymous.SNV <- apply(SNV[1:15], 2, count.nonsyn);
other.mutations <- total.counts - nonsynonymous.SNV;
```

```
# subset the first fifteen samples
barplot.data <- data.frame(
    samples = rep(1:15, 2),
    mutation = c(rep('nonsynonymous', 15), rep('other',15)),
    type = c(rep(1, 15), rep(2,15)),
    values = c(nonsynonymous.SNV, other.mutations),
    sex = rep(patient$sex[1:15], 2),
    stage = rep(patient$stage[1:15], 2),
    msi = rep(patient$msi[1:15], 2)
    );

# Minimal input
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Minimal_Input', fileext = '.tiff'),
    formula = values ~ samples ,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Minimal input',
    # Editing the metadata
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes labels & limits
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Custom_Axes', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Axes labels & limits',
    # Setting axes labels
    xlab.lab = 'Sample',
    ylab.lab = 'Nonsynonymous SNVs',
    # Setting y-axis limits and tick-mark locations
    ylimits = c(0,30),
    yat = seq(0,30,5),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Font size and font face
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Font_Changes', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Font changes',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    # Changing font sizes
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
```

```
    # Changing font type
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Sorting data
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Sorted', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Sorted bars',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Order bars either by 'increasing' or 'decreasing'
    sample.order = 'decreasing',
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Sorting data with horizontal barplot
create.barplot(
    formula = samples ~ values,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Sorted bars',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    xlimits = c(0,30),
    xat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Order bars either by 'increasing' or 'decreasing'
    sample.order = 'decreasing',
    plot.horizontal = TRUE,
    resolution = 100
    )

# Log-Scaled Axis
log.data <- data.frame(
```

```
    x = 10 ** sample(1:15, 5),
    y = LETTERS[1:5]
    );

create.barplot(
    formula = x ~ y,
    data = log.data,
    # Log base 10 scale y-axis
    yat = 'auto.log',
    main = 'Log Scaled',
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colour changes
sex.colours <- replace(as.vector(barplot.data$sex), which(barplot.data$sex == 'male'),'dodgerblue');
sex.colours <- replace(sex.colours, which(barplot.data$sex == 'female'), 'pink');

create.barplot(
    # filename = tempfile(pattern = 'Barplot_Colour_Changes', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Colour changes',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Colour bars based on sex
    col = sex.colours,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Legend
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Legend', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Legend',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
```

```
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        col = sex.colours,
        # Adding legend to explain bar colour-coding
        legend = list(
            inside = list(
                fun = draw.key,
                args = list(
                    key = list(
                        points = list(
                            col = 'black',
                            pch = 22,
                            cex = 3,
                            fill = c('dodgerblue', 'pink')
                            ),
                        text = list(
                            lab = c('Male','Female')
                            ),
                        padding.text = 5,
                        cex = 1
                        )
                    ),
                    # Positioning legend on plot
                    x = 0.75,
                    y = 0.95
                )
            ),
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Grouped barplot
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Grouped', fileext = '.tiff'),
        formula = values ~ samples,
        data = barplot.data,
        main = 'Grouped bar chart',
        xlab.lab = 'Samples',
        ylab.lab = 'Mutations',
        ylimits = c(0,30),
        yat = seq(0,30,5),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        # Setting groups
        groups = mutation,
        col = default.colours(12, is.greyscale = FALSE)[11:12],
        legend = list(
            inside = list(
                fun = draw.key,
```

```
                args = list(
                    key = list(
                        points = list(
                            col = 'black',
                            pch = 22,
                            cex = 2,
                            fill = default.colours(12, is.greyscale = FALSE)[11:12]
                            ),
                        text = list(
                            lab = c('Nonsynonymous SNV','Other SNV')
                            ),
                        padding.text = 3,
                        cex = 1
                        )
                    ),
                x = 0.55,
                y = 0.95
                )
            ),
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Grouped labels
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Grouped_Labels', fileext = '.tiff'),
        formula = values ~ samples,
        data = barplot.data,
        main = 'Grouped labels',
        xlab.lab = 'Samples',
        ylab.lab = 'Mutations',
        ylimits = c(0,30),
        yat = seq(0,30,5),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        # Setting groups
        groups = mutation,
        col = default.colours(12, is.greyscale = FALSE)[11:12],
        # Grouped labels
        xaxis.lab = rep(c('nonsynonymous', 'other'), 15),
        xaxis.rot = 90,
        group.labels = TRUE,
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Stacked barplot
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Stacked', fileext = '.tiff'),
```

```
    formula = values ~ samples,
    data = barplot.data,
    main = 'Stacked bar chart',
    xlab.lab = 'Samples',
    ylab.lab = 'Mutations',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    groups = mutation,
    col = default.colours(12, is.greyscale = FALSE)[11:12],
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 2,
                        # reverse order to match stacked bar order
                        fill = rev(default.colours(12, is.greyscale = FALSE)[11:12])
                        ),
                    text = list(
                        # reverse order to match stacked bar order
                        lab = rev(c('Nonsynonymous SNV','Other SNV'))
                        ),
                    padding.text = 3,
                    cex = 1
                    )
                ),
            x = 0.55,
            y = 0.95
            )
        ),
    # Changing the plot from a grouped plot to a stacked plot
    stack = TRUE,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Panel organization
create.barplot(
   # filename = tempfile(pattern = 'Barplot_Panel_Layout_numeric_conditioning', fileext = '.tiff'),
    # Setting the panel layout
    formula = values ~ samples | type,
    data = barplot.data,
    main = 'Panel layout',
    xlab.lab = 'Samples',
```

```
    ylab.lab = 'Mutations',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

create.barplot(
    # Setting the panel layout
    formula = values ~ samples | mutation,
    data = barplot.data,
    main = 'Panel layout',
    xlab.lab = 'Samples',
    ylab.lab = 'Mutations',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Panel organization 2
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Panel_Layout_2', fileext = '.tiff'),
    formula = values ~ samples | mutation,
    data = barplot.data,
    main = 'Panel layout',
    xlab.lab = 'Samples',
    ylab.lab = 'Mutations',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Adjusting the panel layout
    layout = c(1,2),
    y.spacing = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
```

```
    );

# Covariates
# Note: Covariates can also be created using the create.multiplot function

# set covariate colour schemes
covariate.colours.sex <- as.character(barplot.data$sex);
covariate.colours.sex[covariate.colours.sex == 'male'] <- 'dodgerblue';
covariate.colours.sex[covariate.colours.sex == 'female'] <- 'pink';

covariate.colours.stage <- as.character(barplot.data$stage);
covariate.colours.stage[covariate.colours.stage == 'I'] <- 'plum1';
covariate.colours.stage[covariate.colours.stage == 'II'] <- 'orchid1';
covariate.colours.stage[covariate.colours.stage == 'III'] <- 'orchid3';
covariate.colours.stage[covariate.colours.stage == 'IV'] <- 'orchid4';

covariate.colours.msi <- as.character(barplot.data$msi);
covariate.colours.msi[covariate.colours.msi == 'MSS'] <- 'chartreuse4';
covariate.colours.msi[covariate.colours.msi == 'MSI-High'] <- 'chartreuse2';

# create object to draw covariates
covariates.object <- list(
    rect = list(
        col = 'white',
        fill = covariate.colours.sex,
        lwd = 1.5
        ),
    rect = list(
        col = 'white',
        fill = covariate.colours.stage,
        lwd = 1.5
        ),
    rect = list(
        col = 'white',
        fill = covariate.colours.msi,
        lwd = 1.5
        )
    );

# see BoutrosLab.plotting.general::covariates.grob() for more information
covariate.object.grob <- covariates.grob(
    covariates = covariates.object,
    ord = c(1:15),
    side = 'top',
    size = 0.8
    );

# Create legend to explain covariates
covariates.legends <- list(
    legend = list(
        colours = c('dodgerblue','pink'),
        labels = c('male','female'),
        title = 'Sex',
```

```
        border = 'white'
        ),
    legend = list(
        colours = c('plum1', 'orchid1', 'orchid3', 'orchid4'),
        labels = c('I','II','III','IV'),
        title = 'Stage',
        border = 'white'
        ),
    legend = list(
        colours = c('chartreuse4','chartreuse2'),
        labels = c('MSS','MSI-High'),
        title = 'MSI',
        border = 'white'
        )
    );

# see BoutrosLab.plotting.general::legend.grob() for more information
covariate.legend.grob <- legend.grob(
    legends = covariates.legends,
    title.just = 'left'
    );

create.barplot(
    # filename = tempfile(pattern = 'Barplot_Covariates', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Covariates',
    ylab.lab = 'Mutations',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    yaxis.fontface = 1,
    # removing x-axis formatting to give space to covariates
    xaxis.tck = 0,
    xaxis.lab = rep('',15),
    xaxis.cex = 0,
    # covariates
    legend = list(
        bottom = list(fun = covariate.object.grob),
        right = list(fun = covariate.legend.grob)
        ),
    key = list(
        x = 1,
        y = -0.028,
        text = list(
            lab = c('Sex','Stage','MSI')
            ),
        padding.text = 1
        ),
    bottom.padding = 4,
    description = 'Barplot created by BoutrosLab.plotting.general',
```

```
        resolution = 200
        );

    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Auto_legend', fileext = '.tiff'),
        formula = values ~ samples,
        data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
        main = 'Covariates',
        ylab.lab = 'Mutations',
        ylimits = c(0,30),
        yat = seq(0,30,5),
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        yaxis.fontface = 1,
        # removing x-axis formatting to give space to covariates
        xaxis.tck = 0,
        xaxis.lab = rep('',15),
        xaxis.cex = 0,
        # covariates
        legend = list(
            inside = list(fun = covariate.legend.grob)
            ),
        bottom.padding = 4,
        inside.legend.auto = TRUE,
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Horizontal orientation
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Horizontal', fileext = '.tiff'),
        # switch formula order
        formula = samples ~ values,
        data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
        main = 'Plot horizontally',
        # Adjusting the panel layout
        plot.horizontal = TRUE,
        # covariates
        legend = list(
            inside = list(fun = covariate.legend.grob)
            ),
        inside.legend.auto = TRUE,
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Change bar thickness and add text labels
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Text_Labels', fileext = '.tiff'),
        # switch formula order
        formula = samples ~ values,
        data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
```

```
    main = 'Text labels and thin bars',
    # Adjusting the panel layout
    plot.horizontal = TRUE,
    box.ratio = 0.6,
    add.text = TRUE,
    text.x = 27.75,
    text.y = 1:15,
    text.labels = barplot.data[barplot.data$mutation == 'nonsynonymous','values'],
    text.cex = 0.8,
    text.fontface = 'italic',
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Error bars
error.data <- data.frame(
    genes = rownames(microarray)[1:15],
    values = apply(microarray[1:15,1:58], 1, mean),
    error = apply(microarray[1:15,1:58], 1, sd)
    );

create.barplot(
    # filename = tempfile(pattern = 'Barplot_Error_Bars', fileext = '.tiff'),
    # needs sequential x-axis
    formula = values ~ 1:15,
    data = error.data,
    y.error.up = error.data$error,
    xaxis.lab = error.data$genes,
    main = 'Error bars',
    xlab.lab = 'Gene',
    ylab.lab = 'Change in Expression',
    ylimits = c(0,14),
    yat = seq(0,14,2),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.rot = 45,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


create.barplot(
    # filename = tempfile(pattern = 'Barplot_Error_Bars_Horizontal', fileext = '.tiff'),
    # needs sequential x-axis
    formula = values ~ 1:15,
    data = error.data,
    y.error.up = error.data$error,
    yaxis.lab = error.data$genes,
    plot.horizontal = TRUE,
```

```
    main = 'Error bars',
    xlab.lab = 'Gene',
    ylab.lab = 'Change in Expression',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.rot = 45,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Grid lines
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Gridlines', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Gridlines',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Grid lines
    add.grid = TRUE,
    xgrid.at = seq(0,15,2),
col = sex.colours,
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 3,
                        fill = c('dodgerblue', 'pink')
                        ),
                    text = list(
                        lab = c('Male','Female')
                        ),
                    padding.text = 5,
                    cex = 1
                    )
                ),
                # Positioning legend on plot
```

```
                x = 0.75,
                y = 0.95
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Grid lines 2
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Gridlines_GreyBG', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Gridlines & grey background',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Grid lines
    background.col = 'grey85',
    add.grid = TRUE,
    xgrid.at = seq(0,15,2),
    col = sex.colours,
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 3,
                        fill = c('dodgerblue', 'pink')
                        ),
                    text = list(
                        lab = c('Male','Female')
                        ),
                    padding.text = 5,
                    cex = 1
                    )
                ),
                # Positioning legend on plot
                x = 0.75,
                y = 0.95
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
```

```
        resolution = 200
        );

    # Labels
    create.barplot(
        # filename = tempfile(pattern = 'Barplot_Labels', fileext = '.tiff'),
        formula = values ~ samples,
        data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
        main = 'Labels',
        xlab.lab = 'Samples',
        ylab.lab = 'Nonsynonymous SNVs',
        ylimits = c(0,30),
        yat = seq(0,30,5),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        # Labels
        text.above.bars = list(
            labels = c('*','27','15','*'),
            padding = 0.75,
            bar.locations = c(1, 3, 12, 14),
            rotation = 0
            ),
        col = sex.colours,
        legend = list(
            inside = list(
                fun = draw.key,
                args = list(
                    key = list(
                        points = list(
                            col = 'black',
                            pch = 22,
                            cex = 3,
                            fill = c('dodgerblue', 'pink')
                            ),
                        text = list(
                            lab = c('Male','Female')
                            ),
                        padding.text = 5,
                        cex = 1
                        )
                    ),
                    # Positioning legend on plot
                    x = 0.75,
                    y = 0.95
                )
            ),
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 200
        );
```

```
# lines
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Lines', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Lines',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Lines
    sample.order = 'increasing',
    line.func = function(x) {0.1*x**2},
    line.from = 0,
    line.to = 16,
    line.col = 'darkgrey',
    abline.h = 10,
    abline.col = 'red',
    col = sex.colours,
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 3,
                        fill = c('dodgerblue', 'pink')
                        ),
                    text = list(
                        lab = c('Male','Female')
                        ),
                    padding.text = 5,
                    cex = 1
                    )
                ),
                # Positioning legend on plot
                x = 0.75,
                y = 0.95
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# Background rectangle
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Bg_Rectangle', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Background rectangle',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    sample.order = 'increasing',
    # Background rectangle
    add.rectangle = TRUE,
    xleft.rectangle = seq(0.5, 14.5, 2),
    ybottom.rectangle = 0,
    xright.rectangle = seq(1.5, 15.5, 2),
    ytop.rectangle = 30,
    col.rectangle = 'lightgrey',
    col = sex.colours,
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 3,
                        fill = c('dodgerblue', 'pink')
                        ),
                    text = list(
                        lab = c('Male','Female')
                        ),
                    padding.text = 5,
                    cex = 1
                    )
                ),
                # Positioning legend on plot
                x = 0.75,
                y = 0.95
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Raster
```

```
create.barplot(
    # filename = tempfile(pattern = 'Barplot_with_raster', fileext = '.tiff'),
    formula = values ~ samples,
    data = barplot.data[barplot.data$mutation == 'nonsynonymous',],
    main = 'Raster fill',
    xlab.lab = 'Samples',
    ylab.lab = 'Nonsynonymous SNVs',
    ylimits = c(0,30),
    yat = seq(0,30,5),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # filling bars with raster
    raster = 1:10/10,
    raster.just = 'bottom',
    description = 'Description of image here',
    resolution = 200
    );

# Nature format
create.barplot(
    # filename = tempfile(pattern = 'Barplot_Nature_style', fileext = '.tiff'),
    formula = x ~ y,
    data = simple.data,
    yat = seq(0,16,2),
    main = 'Nature style',

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    resolution = 200
    );

# Left Justified Example
create.barplot(
   # filename = tempfile(pattern = 'Barplot_TwoTopLabelsLeftJustified', fileext = '.tiff'),
    formula = x ~ y,
    data = simple.data,
    yat = seq(0,16,2),
    ylab.label = NULL,
    # set top label details
    xlab.top.label = 'Sample Label',
    xlab.top.cex = 1.5,
    xlab.top.x = -0.125,
```

```
    xlab.top.y = 0.5,
    xlab.top.just = 'left',
    # set main label details
    main = 'Sample Main',
    main.just = 'left',
    main.x = 0,
    main.y = 0.6,
    top.padding = 1,
    resolution = 200
    );
```

`create.boxplot` *Make a boxplot*

Description

Takes a data.frame and creates a boxplot

Usage

```
create.boxplot(
formula,
data,
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
add.stripplot = FALSE,
jitter.factor = 1,
jitter.amount = NULL,
points.pch = 19,
points.col = 'darkgrey',
points.cex = 0.5,
points.alpha = 1,
abline.h = NULL,
abline.v = NULL,
abline.lty = NULL,
abline.lwd = NULL,
abline.col = 'black',
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
```

```
    alpha.rectangle = 1,
    box.ratio = 1,
    col = 'transparent',
    alpha = 1,
    border.col = 'black',
    symbol.cex = 0.8,
    lwd = 1,
    outliers = TRUE,
    sample.order = 'none',
    order.by = 'median',
    xlab.label = tail(sub('~', '', formula[-2]), 1),
    ylab.label = tail(sub('~', '', formula[-3]), 1),
    xlab.cex = 2,
    ylab.cex = 2,
    xlab.col = 'black',
    ylab.col = 'black',
    xlab.top.label = NULL,
    xlab.top.cex = 2,
    xlab.top.col = 'black',
    xlab.top.just = 'center',
    xlab.top.x = 0.5,
    xlab.top.y = 0,
    xlimits = NULL,
    ylimits = NULL,
    xat = TRUE,
    yat = TRUE,
    xaxis.lab = TRUE,
    yaxis.lab = TRUE,
    xaxis.cex = 1.5,
    yaxis.cex = 1.5,
    xaxis.col = 'black',
    yaxis.col = 'black',
    xaxis.fontface = 'bold',
    yaxis.fontface = 'bold',
    xaxis.rot = 0,
    yaxis.rot = 0,
    xaxis.tck = c(1,0),
    yaxis.tck = 1,
    layout = NULL,
    as.table = FALSE,
    x.spacing = 0,
    y.spacing = 0,
    x.relation = 'same',
    y.relation = 'same',
    top.padding = 0.5,
    bottom.padding = 2,
    right.padding = 1,
    left.padding = 2,
```

```
ylab.axis.padding = 0,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.anchor = 'centre',
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
key = NULL,
legend = NULL,
strip.col = 'white',
strip.cex = 1,
strip.fontface = 'bold',
line.func = NULL,
line.from = 0,
line.to = 0,
line.col = 'transparent',
line.infront = TRUE,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE,
disable.factor.sorting = FALSE
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame. Transforming data within formula is not compatible with automatic scaling with ‘xat‘ or ‘yat‘.
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 3
`add.stripplot`	logical whether to plot all points, defaults to FALSE
`jitter.factor`	Numeric value to apply to jitter, default is 1
`jitter.amount`	Numeric; amount of noise to add, default is NULL

`points.pch`	pch value to use for stripplot
`points.col`	colour(s) to use for stripplot (either a single colour or a vector)
`points.cex`	cex value to use for stripplot
`points.alpha`	alpha value to use for stripplot
`abline.h`	Specify the horizontal superimpose line
`abline.v`	Specify the vertical superimpose line
`abline.lty`	Specify the superimpose line type
`abline.lwd`	Specify the superimpose line width
`abline.col`	Specify the superimpose line colour (defaults to black)
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour of the rectangle to be drawn
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`box.ratio`	ability to change the box width, defaults to 1
`col`	The colour to fill the interior of the boxplot, defaults to white
`alpha`	The alpha of the interior boxplot colour specified in ‘col‘. Defaults to 1 (opaque)
`border.col`	Colour of the boxplot, defaults to black
`symbol.cex`	Size of the boxplot outlier-symbol
`lwd`	Line width, defaults to 1
`outliers`	logical whether to plot outliers, defaults to TRUE
`sample.order`	String specifying how samples should be ordered. Either none, increasing, or decreasing.
`order.by`	A string specifying what the sample order should be ordered by, either max, min, median or mean
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label

`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits
`ylimits`	Two-element vector giving the y-axis limits
`xat`	Accepts a vector listing where x-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes x-axis tick locations, labels, and data values dependent on given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`yat`	Accepts a vector listing where y-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes y-axis tick locations, labels, and data values dependent on given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with xat will overwrite user input. Set to NULL to remove x-axis labels.
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with yat will overwrite user input. Set to NULL to remove y-axis labels.
`xaxis.cex`	Size of x-axis tick labels, defaults to 2
`yaxis.cex`	Size of y-axis tick labels, defaults to 2
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1 (bottom) and 0 (top)
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
.	
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down

`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`top.padding`	A number specifying the distance to the top margin, defaults to 0.5
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 2
`right.padding`	A number specifying the distance to the right margin, defaults to 1
`left.padding`	A number specifying the distance to the left margin, defaults to 2
`ylab.axis.padding`	A number specifying the distance of y-axis label to the y-axis, defaults to 0

,

`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text. If the formula contains group, the length of this argument should match with the number of groups.
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.anchor`	Part of text that should be anchored to x/y coordinates. Defaults to ’centre’. Use ’left’ or ’right’ to left or right-align text.
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`key`	Add a key to the plot. See xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`strip.fontface`	Strip title fontface, defaults to bold
`line.func`	Function for the line that should be drawn on top of plot
`line.from`	The starting point of the line on the plot
`line.to`	The ending point of the line on the plot
`line.col`	Colour of the line on the plot
`line.infront`	Should the line appear in front of the plot or not
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE

`description`	Description of image/plot; default NULL.
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`disable.factor.sorting`	Disable barplot auto sorting factors alphabetically/numerically

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Maud H.W. Starmans

See Also

`bwplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = rnorm(1000),
    y = rep('A',1000)
    );

create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Simple', fileext = '.tiff'),
    formula = y ~ x,
    data = simple.data,
    main = 'Simple',
```

```
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# add stripplot behind boxplot
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_with_Stripplot', fileext = '.tiff'),
    formula = y ~ x,
    data = simple.data,
    main = 'With Stripplot',
    add.stripplot = TRUE,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Multi-coloured stripplot
strip.data <- data.frame(
    score = c(rnorm(30, 15, 3), rnorm(50, 20, 4)),
    sex = sample(c('male', 'female'), 80, replace = TRUE),
    gene = sample(c('a', 'b'), 80, replace = TRUE)
    );

create.boxplot(
    filename = NULL,
    formula = score ~ sex | gene,
    data = strip.data,
    main = 'Multi-Coloured Stripplot',
    add.stripplot = TRUE,
    points.col = c('pink', 'dodgerblue'),
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# format data
reformatted.data <- data.frame(
    x = as.vector(t(microarray[1:10,1:58])),
    y = as.factor(rep(rownames(microarray[1:10,1:58]),each = 58)),
    z = sample(1:10, 580, replace = TRUE)
    );

# Minimal Input
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Minimal_Input', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Minimal input',
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Minimal Input
create.boxplot(
```

```
  # filename = tempfile(pattern = 'Boxplot_Disable_Factor_Sorting_Input', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'No Factor Sorting',
    disable.factor.sorting = TRUE,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Axes and labels
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Axes_Labels', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Axes & labels',
    # Adjusting axes size
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # Adding y-axis label
    ylab.label = 'Gene',
    # setting axes limits
    xlimits = c(0,13),
    xat = seq(0,12,2),
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Sorting
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Sorted', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Sorting',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Reordered by median
    sample.order = 'increasing',
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colour change
sex.colour <- as.character(patient$sex);
sex.colour[sex.colour == 'male'] <- 'dodgerblue';
sex.colour[sex.colour == 'female'] <- 'pink';
```

```
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Colour_Change', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Colour change',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Colour change
    col = sex.colour,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Remove y-axis labels
create.boxplot(
    formula = y ~ x,
    data = reformatted.data,
    main = 'Remove y-axis labels',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    yaxis.lab = NULL, # Remove labels with NULL
    # Colour change
    col = sex.colour,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Log Scaled Axis
log.data <- data.frame(
    x = 10 ** rnorm(1000, 5, 2),
    y = rep('A',1000)
    );

create.boxplot(
    formula = x ~ y,
    data = log.data,
    # Log base 10 scale y axis
    yat = 'auto.log',
    main = 'Log Scale',
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );
```

```
# Legend
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Legend', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Legend',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    col = sex.colour,
    # legend
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 1.5,
                        fill = c('dodgerblue','pink')
                        ),
                    text = list(
                        lab = c('male','female')
                        ),
                    cex = 1
                    )
                ),
            x = 0.03,
            y = 0.97,
            corner = c(0,1),
            draw = FALSE
            )
        ),
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Orientation
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Orientation', fileext = '.tiff'),
    # switch the order
    formula = x ~ y,
    data = reformatted.data,
    main = 'Orientation',
    xaxis.cex = 1,
    yaxis.cex = 1,
    # adjust the axes
```

```
    ylimits = c(0,13),
    yat = seq(0,12,2),
    # rotate the labels
    xaxis.rot = 90,
    xlab.label = 'Gene',
    xlab.cex = 1.5,
    col = sex.colour,
    # legend
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = 'black',
                        pch = 22,
                        cex = 1.5,
                        fill = c('dodgerblue','pink')
                        ),
                    text = list(
                        lab = c('male','female')
                        ),
                    cex = 1
                    )
                ),
            x = 0.23,
            y = 0.97,
            corner = c(0,1),
            draw = FALSE
            )
        ),
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Background rectangle
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_BG_Rect', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Bg rectangle',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # draw rectangle
    add.rectangle = TRUE,
    xleft.rectangle = 0,
    xright.rectangle = 13,
    ybottom.rectangle = seq(0.5, 8.5, 2),
```

```
        ytop.rectangle = seq(1.5, 9.5, 2),
        col.rectangle = 'grey',
        alpha.rectangle = 0.5,
        col = sex.colour,
        # legend
        legend = list(
            inside = list(
                fun = draw.key,
                args = list(
                    key = list(
                        points = list(
                            col = 'black',
                            pch = 22,
                            cex = 1.5,
                            fill = c('dodgerblue','pink')
                            ),
                        text = list(
                            lab = c('male','female')
                            ),
                        cex = 1
                        )
                    ),
                x = 0.03,
                y = 0.97,
                corner = c(0,1),
                draw = FALSE
                )
            ),
        description = 'Boxplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Line
    create.boxplot(
        # filename = tempfile(pattern = 'Boxplot_Line', fileext = '.tiff'),
        formula = y ~ x,
        data = reformatted.data,
        main = 'Line',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        ylab.label = 'Gene',
        xlimits = c(0,13),
        xat = seq(0,12,2),
        # draw line
        line.func = function(x){c(0.5, 10.5)},
        line.from = 11,
        line.to = 11,
        line.col = 'grey',
        description = 'Boxplot created by BoutrosLab.plotting.general',
        resolution = 200
        );
```

```
# Panel Organization
create.boxplot(
   # filename = tempfile(pattern = 'Boxplot_Panels_numeric_conditioning', fileext = '.tiff'),
    formula = ~ x | z,
    data = reformatted.data,
    main = 'Panels',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Setting up the layout
    layout = c(2,5),
    x.relation = 'free',
    x.spacing = 1,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

create.boxplot(
   # filename = tempfile(pattern = 'Boxplot_Panels_factor_conditioning', fileext = '.tiff'),
    formula = ~ x | y,
    data = reformatted.data,
    main = 'Panels',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Setting up the layout
    layout = c(2,5),
    x.relation = 'free',
    x.spacing = 1,
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature format
create.boxplot(
    # filename = tempfile(pattern = 'Boxplot_Nature_style', fileext = '.tiff'),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Nature style',
    xaxis.cex = 1,
    yaxis.cex = 1,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),
```

```
    resolution = 1200
    );

# Sorting by mean and multiple filenames
create.boxplot(
    filename = c(
        tempfile(pattern = 'Boxplot_Sorted1', fileext = '.tiff'),
        tempfile(pattern = 'Boxplot_Sorted2', fileext = '.tiff')
        ),
    formula = y ~ x,
    data = reformatted.data,
    main = 'Sorting',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    ylab.label = 'Gene',
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Reordered by median
    sample.order = 'increasing',
    order.by = 'mean',
    description = 'Boxplot created by BoutrosLab.plotting.general',
    resolution = 200
    );


# Adding text to plot
# Generate normally distributed variables with two different means
set.seed(779);
groupA <- rnorm(n = 100, mean = 10, sd = 2);
groupB <- rnorm(n = 134, mean = 10.5, sd = 2);

# Create data frame for plotting
to.plot <- data.frame(
y = rep(
c('1', '2'),
times = c(100, 134)
),
x = c(groupA, groupB)
);

# Get difference between means
diff.mean <- round(mean(groupB) - mean(groupA), 2);

# Plot and display difference
create.boxplot(
formula = x ~ y,
# filename = tempfile(pattern = 'boxplot_with_text', fileext = '.tiff'),
data = to.plot,
add.stripplot = TRUE,
add.text = TRUE,
text.labels = bquote(mu[B] - mu[A] == .(diff.mean)),
```

```
text.x = 2.1,
text.y = 15.3,
text.col = 'black',
text.cex = 1.5,
text.fontface = 'bold',
ylimits = c(
min(to.plot$x) - abs(min(to.plot$x) * 0.1),
max(to.plot$x) + abs(max(to.plot$x) * 0.1)
),
resolution = 200
    );
```

`create.colourkey` *Create Colourkey*

Description

A function for generating and placing a colour key. Good for use in multiplots when a smaller colour key is desired.

Usage

```
create.colourkey(
x,
scale.data = FALSE,
colour.scheme = c(),
total.colours = 99,
colour.centering.value = 0,
colour.alpha = 1,
fill.colour = 'darkgray',
at = NULL,
colourkey.labels.at = NULL,
colourkey.labels = colourkey.labels.at,
colourkey.labels.cex = 1,
placement = NULL
);
```

Arguments

`x`	Either a data-frame or a matrix from which the heatmap was created
`scale.data`	Was the data for the heatmap scaled? Defaults to FALSE.
`colour.scheme`	Heatmap colouring. Accepts old-style themes, or a vector of either two or three colours that are gradiated to create the final palette.
`total.colours`	Total number of colours to plot.
`colour.centering.value`	The center of the colour-map.

`colour.alpha`	Bias to be added to colour selection (uses x^colour.alpha in mapping).
`fill.colour`	The background fill (only exposed where missing values are present.
`at`	A vector specifying the breakpoints along the range of x.
`colourkey.labels.at`	A vector specifying the tick-positions on the colourkey.
`colourkey.labels`	A vector specifying tick-labels of the colourkey
`colourkey.labels.cex`	Size of colourkey labels. Defaults to 1
`placement`	Location and size of the colourkey.

Value

Returns a key in the format specified in the xyplot documentation.

Author(s)

Stephenie Prokopec

See Also

xyplot, plotmath

Examples

```
set.seed(1234567890);
x <- outer(-5:5, -5:5, '*') + matrix(nrow = 11, ncol = 11, data = runif(11 * 11));
colnames(x) <- paste('col', 1:11, sep = '-');
rownames(x) <- paste('row', 1:11, sep = '-');

y <- as.data.frame(x);
y$mean <- apply(x,1,mean);

# example of a simple multiplot with colourkey
heatmap1 <- create.heatmap(
    x = t(x),
    filename = NULL,
    clustering.method = 'none',
    scale.data = FALSE,
    yaxis.lab = NA,
    print.colour.key = FALSE,
    colour.scheme = c('chartreuse3', 'white', 'blue'),
    at = seq(-25, 25, 0.01)
    );

barplot1 <- create.barplot(
    1:nrow(y) ~ mean,
    y,
    plot.horizontal = TRUE
    );
```

```
create.multiplot(
    plot.objects = list(heatmap1, barplot1),
    # filename = tempfile(pattern = 'multiplot_with_colourkey', fileext = '.tiff'),
    plot.layout = c(2,1),
    panel.widths = c(2,1),
    yat = list(1:nrow(y), NULL),
    yaxis.labels = rownames(y),
    xlimits = list(NULL, c(0,1)),
    xat = list(NULL, seq(0,1,0.5)),
    xaxis.labels = list(NULL, seq(0,1,0.5)),
    x.spacing = 0,
    print.new.legend = TRUE,
    legend = list(
        inside = list(
            fun = BoutrosLab.plotting.general::create.colourkey(
                x = x,
                colour.scheme = c('chartreuse3', 'white', 'blue'),
                at = seq(-25, 25, 0.01),
                colourkey.labels.at = c(-25, 0, 25),
                placement = viewport(just = 'left', x = 0.55, y = -0.55, width = 0.5)
                )
            )
        ),
    bottom.padding = 4,
    width = 10,
    height = 8,
    resolution = 500
    );
```

`create.dendrogram` *Generate a dendrogram*

Description

Takes a matrix and creates a row-wise or column-wise dendrogram

Usage

```
create.dendrogram(
x,
clustering.method = 'diana',
cluster.dimension = 'col',
distance.method = 'correlation',
cor.method = 'pearson',
force.clustering = FALSE,
same.as.matrix = FALSE
);
```

Arguments

`x`	A matrix that is used to create the dendrogram
`clustering.method`	Method used to cluster the records (can not be none). Accepts all agglomerative clustering methods available in hclust, plus "diana" (which is divisive).
`cluster.dimension`	Should clustering be performed on the rows or columns of x?
`distance.method`	Method name of the distance measure to be used for clustering. Defaults to "correlation". Other supported methods are same as in ?dist. Also supports "jaccard" which is useful for clustering categorical variables.
`cor.method`	The method used for calculating correlation. Defaults to "pearson"
`force.clustering`	Binary to over-ride the control that prevents clustering of too-large matrices
`same.as.matrix`	Prevents the flipping of the matrix that the function normally does

Value

Returns an object of the dendrogram class corresponding to the row-wise or column-wise dendrogram for x

Author(s)

Lauren Chong

Examples

```
# create temp data
x <- outer(-5:5, -5:5, '*') + matrix(nrow = 11, ncol = 11, data = runif(11 * 11));
colnames(x) <- paste('col', 1:11, sep = '-');
rownames(x) <- paste('row', 1:11, sep = '-');

# example of generating a column-wise dendrogram using default values
create.dendrogram(
    x = x
    );

# example of generating a column-wise dendrogram using different distance and clustering methods
create.dendrogram(
    x = x,
    clustering.method = 'median',
    cluster.dimension = 'cols',
    distance.method = 'euclidean'
    );

# generate row-wise dendrogram using default distance and clustering methods
create.dendrogram(
    x = x,
    cluster.dimension = 'row'
```

```
    );

# generate row-wise dendrogram using different distance and clustering methods
create.dendrogram(
    x = x,
    clustering.method = 'ward',
    cluster.dimension = 'rows',
    distance.method = 'manhattan'
    );
```

`create.densityplot` *Make a density plot*

Description

Takes a list of vectors and creates a density-plot with each vector as a separate curve

Usage

```
create.densityplot(
x,
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = NULL,
ylab.label = 'Density',
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
type = 'l',
lty = 'solid',
cex = 0.75,
pch = 19,
col = 'black',
lwd = 2,
bandwidth = 'nrd0',
bandwidth.adjust = 1,
xlimits = NULL,
```

```
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.lab = NA,
yaxis.lab = NA,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.tck = 1,
yaxis.tck = 1,
xgrid.at = xat,
ygrid.at = yat,
key = list(text = list(lab = c(''))),
legend = NULL,
top.padding = 0.1,
bottom.padding = 0.7,
left.padding = 0.5,
right.padding = 0.1,
add.axes = FALSE,
abline.h = NULL,
abline.v = NULL,
abline.lty = NULL,
abline.lwd = NULL,
abline.col = 'black',
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.anchor = "centre",
text.col = "black",
text.cex = 1,
        text.fontface = "bold",
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
```

```
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE,
inside.legend.auto = FALSE
);
```

Arguments

`x`	A list of vectors, each of which will be plotted as a separate curve in the final plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 2
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis, defaults to “Density”
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`type`	Plot type
`lty`	Line type
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`col`	Point/line colour
`lwd`	Thickness of width of any best-fit lines
`bandwidth`	Smoothing bandwidth, or character string giving rule to choose bandwidth (’nrd0’, ’nrd’, ’ucv’, ’bcv’, ’sj’, or ’sj-ste’). Passed to base R function density.
`bandwidth.adjust`	Adjustment parameter for the bandwidth (bandwidth used is bandwidth*bandwidth.adjust). Makes it easy to specify bandwidth as a proportion of the default.

`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis tick labels, defaults to 1
`yaxis.cex`	Size of x-axis tick labels, defaults to 1
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xgrid.at`	Vector listing where the x-axis grid lines should be drawn, defaults to xat
`ygrid.at`	Vector listing where the y-axis grid lines should be drawn, defaults to yat
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`top.padding`	A number giving the top padding in multiples of the lattice default
`bottom.padding`	A number giving the bottom padding in multiples of the lattice default
`left.padding`	A number giving the left padding in multiples of the lattice default
`right.padding`	A number giving the right padding in multiples of the lattice default
`add.axes`	Allow axis lines to be turned on or off
`abline.h`	Specify the superimposed horizontal line(s)
`abline.v`	Specify the superimposed vertical line(s)
`abline.lty`	Specify the superimposed line type
`abline.lwd`	Specify the superimposed line width
`abline.col`	Specify the superimposed line colour (defaults to black)
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn

`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.anchor`	Part of text that should be anchored to x/y coordinates. Defaults to 'centre'. Use 'left' or 'right' to left or right-align text.
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = rnorm(1000),
    y = rnorm(1000, mean = 3, sd = 3)
    );

create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Simple', fileext = '.tiff'),
    x = simple.data,
    main = 'Simple',
    description = 'Barplot created by BoutrosLab.plotting.general'
    );

# format data
format.data <- microarray[1:3,1:58];
format.data <- as.data.frame(t(format.data));

# Minimal Input
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Minimal_Input', fileext = '.tiff'),
    x = format.data,
    main = 'Minimal input',
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Line type
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Line_Type', fileext = '.tiff'),
    x = format.data,
    main = 'Line type',
    # Line type
    lty = c('solid','dashed','dotted'),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Axes & Labels
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Axes_Labels', fileext = '.tiff'),
    x = format.data,
    main = 'Axes & labels',
```

```
    lty = c('solid','dashed','dotted'),
    # Axes & Labels
    ylimits = c(-0.1, 2.5),
    ylab.cex = 1.5,
    xat = seq(0, 13, 1),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 50
    );


# Colour change & Legend
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Colour_Legend', fileext = '.tiff'),
    x = format.data,
    main = 'Colour & legend',
    lty = c('solid','dashed','dotted'),
    ylimits = c(-0.1, 2.5),
    ylab.cex = 1.5,
    # Colours
    col = default.colours(3),
    # Legend
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 21,
                        cex = 1.5,
                        fill = default.colours(3)
                        ),
                    text = list(
                        lab = colnames(format.data)
                        ),
                    padding.text = c(0,5,0),
                    cex = 1
                    )
                ),
            x = 0.65,
            y = 0.97,
            draw = FALSE
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Correlation key
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Correlation_Key', fileext = '.tiff'),
    x = format.data[,1:2],
    main = 'Correlation key',
```

```
    lty = c('solid','dotted'),
    ylimits = c(-0.1, 2.5),
    ylab.cex = 1.5,
    col = default.colours(2),
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(2),
                        pch = 21,
                        cex = 1.5,
                        fill = default.colours(2)
                        ),
                    text = list(
                        lab = colnames(format.data)[1:2]
                        ),
                    padding.text = c(0,5,0),
                    cex = 1
                    )
                ),
            x = 0.65,
            y = 0.97,
            draw = FALSE
            ),
        # Correlation key accepts two vectors
        inside = list(
            fun = draw.key,
            args = list(
                key = get.corr.key(
                    x = as.numeric(format.data[,1]),
                    y = as.numeric(format.data[,2]),
                    label.items = c('pearson','beta1'),
                    alpha.background = 1,
                    key.cex = 1.2
                    )
                ),
            x = 0.65,
            y = 0.85,
            corner = c(0,1)
            )
        ),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Gridlines
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Gridlines_1', fileext = '.tiff'),
    x = format.data,
    main = 'Gridlines',
    lty = c('solid','dashed','dotted'),
```

```
    ylimits = c(-0.1, 2.5),
    ylab.cex = 1.5,
    col = default.colours(3),
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 21,
                        cex = 1.5,
                        fill = default.colours(3)
                        ),
                    text = list(
                        lab = colnames(format.data)
                        ),
                    padding.text = c(0,5,0),
                    cex = 1
                    )
                ),
            x = 0.65,
            y = 0.97,
            draw = FALSE
            )
        ),
    # Grid lines
    type = c('l','g'),
    description = 'Barplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Gridlines
create.densityplot(
    # filename = tempfile(pattern = 'Densityplot_Gridlines_2', fileext = '.tiff'),
    x = format.data,
    main = 'Gridlines',
    lty = c('solid','dashed','dotted'),
    ylimits = c(-0.1, 2.5),
    ylab.cex = 1.5,
    col = default.colours(3),
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 21,
                        cex = 1.5,
                        fill = default.colours(3)
                        ),
                    text = list(
```

```
                            lab = colnames(format.data)
                            ),
                        padding.text = c(0,5,0),
                        cex = 1
                        )
                    ),
                x = 0.65,
                y = 0.97,
                draw = FALSE
                )
            ),
        # Grid lines
        type = c('l','g'),
        xgrid.at = seq(0,14,1),
        ygrid.at = seq(0,2.5,0.25),
        description = 'Barplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Nature style
    create.densityplot(
        # filename = tempfile(pattern = 'Densityplot_Nature_style', fileext = '.tiff'),
        x = format.data,
        main = 'Nature style',
        lty = c('solid','dashed','dotted'),
        ylimits = c(-0.1, 2.5),
        ylab.cex = 1.5,
        xlab.cex = 1.5,
        col = default.colours(3),
        legend = list(
            inside = list(
                fun = draw.key,
                args = list(
                    key = list(
                        points = list(
                            col = default.colours(3),
                            pch = 21,
                            cex = 1.5,
                            fill = default.colours(3)
                            ),
                        text = list(
                            lab = colnames(format.data)
                            ),
                        padding.text = c(0,5,0),
                        cex = 1
                        )
                    ),
                x = 0.65,
                y = 0.97,
                draw = FALSE
                )
            ),
        # Grid lines
```

```
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),
    resolution = 200
    );
```

`create.dotmap` *Make a dotmap with coloured background*

Description

Takes two data.frames and creates a dotmap with a coloured background. A dotmap is an ordered array of evenly-spaced dots whose size and colour can be user-specified to represent characteristics. For example, size gives the absolute magnitude of the correlation and colour gives the sign of the correlation. The coloured background may indicate p-values.

Usage

```
create.dotmap(
x,
bg.data = NULL,
filename = NULL,
main = NULL,
main.just = "center",
main.x = 0.5,
main.y = 0.5,
pch = 19,
pch.border.col = 'black',
add.grid = TRUE,
xaxis.lab = colnames(x),
yaxis.lab = rownames(x),
xaxis.rot = 0,
yaxis.rot = 0,
main.cex = 3,
xlab.cex = 2,
ylab.cex = 2,
xlab.label = NULL,
ylab.label = NULL,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
```

```
xlab.top.just = "center",
xlab.top.x = 0.5,
xlab.top.y = 0,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.tck = 1,
yaxis.tck = 1,
axis.top = 1,
axis.bottom = 1,
axis.left = 1,
axis.right = 1,
top.padding = 0.1,
bottom.padding = 0.7,
right.padding = 0.1,
left.padding = 0.5,
key.ylab.padding = 0.1,
key = list(text = list(lab = c(''))),
legend = NULL,
col.lwd = 1.5,
row.lwd = 1.5,
spot.size.function = 'default',
spot.colour.function = 'default',
na.spot.size = 7,
na.pch = 4,
na.spot.size.colour = 'black',
grid.colour = NULL,
colour.scheme = 'white',
total.colours = 99,
at = NULL,
colour.centering.value = 0,
colourkey = FALSE,
colourkey.labels.at = NULL,
colourkey.labels = NULL,
colourkey.cex = 1,
colour.alpha = 1,
bg.alpha = 0.5,
fill.colour = 'white',
key.top = 0.1,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
col.colour = 'black',
row.colour = 'black',
description = 'Created with BoutrosLab.plotting.general',
```

```
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
border.rectangle = NULL,
lwd.rectangle = NULL,
alpha.rectangle = 1,
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
dot.colour.scheme = NULL,
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE,
remove.symmetric = FALSE,
        lwd = 2
);
```

Arguments

`x`	An unstacked data.frame to plot the dotmap
`bg.data`	An unstacked data.frame to plot the background, of the same size as “x”. Column names specified here may be arbitrary: they are not used in the plot.
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`pch`	Plotting character
`pch.border.col`	Colour of the dot border if using pch = 21:25
`add.grid`	Should a grid of black-lines separating each column/row be added?
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`main.cex`	Size of text for the main title, defaults to 2

`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xaxis.lab`	Vector listing x-axis tick labels, defaults to colnames(x)
`yaxis.lab`	Vector listing y-axis tick labels, defaults to rownames(x)
`xaxis.cex`	Size of x-axis tick labels, defaults to 1.2
`yaxis.cex`	Size of y-axis tick labels, defaults to 1.5
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`axis.top`	Specifies the padding on the top of the plot
`axis.bottom`	Specifies the padding on the bottom of the plot
`axis.left`	Specifies the padding on the left of the plot
`axis.right`	Specifies the padding on the right of the plot
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.1
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`key.ylab.padding`	a number specifying distance between key and left label
`key`	A list giving the key (legend). The default suppresses drawing. If the key has a “space” component then extra space will be cleared on that side of the plot for the key
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`col.lwd`	Thickness of column grid lines
`row.lwd`	Thickness of row grid lines
`spot.size.function`	The function that translates values in x into dotmap spot-size. The default is 0.1 + (2 * abs(x))
`spot.colour.function`	The function that translates values in x into dotmap spot-colour. The default gives negative values blue, positive values red, and zero white. Parameter also accepts ’columns’ and ’rows’, which groups the dot colours by columns or rows (not both), respectively. For column/row grouping, there are 12 unique colours and these colours will start to repeat once there are more than 12 columns/rows.
`na.spot.size`	The size for plotting character for NA cells. Defaults to 7.
`na.pch`	The type of plotting character to represent NA cells. Defaults to 4 (’X’).

`na.spot.size.colour`	Colour for plotting character representing NA cells. Defaults to black.
`grid.colour`	The colour for the grid lines. DEPRECATED
`colour.scheme`	Background colouring. Accepts a vector of colours. Vectors of two or three colours are gradiated to create the final palette. Defaults to “white”.
`total.colours`	Total number of colours to plot for the Background colours
`at`	A vector specifying the breakpoints along the range of bg; each interval specified by these breakpoints are assigned to a colour from the palette. Defaults to NULL, which corresponds to the range of bg being divided into total.colours equally spaced intervals. If bg has values outside of the range specified by “at”, those values are shown with colours corresponding to the extreme ends of the colour spectrum and a warning is given.
`colour.centering.value`	What should be the center of the background key
`colourkey`	Determines if the colour key should be added or not and sets up its formatting. Defaults to FALSE.
`colourkey.labels.at`	A vector specifying the tick-positions on the background colourkey
`colourkey.labels`	A vector specifying tick-labels of the background colourkey
`colourkey.cex`	Size of the background colourkey label text
`colour.alpha`	Bias to be added to background colour selection (uses x^colour.alpha in mapping)
`bg.alpha`	The alpha value of the background colours, defaults to 0.5 so that the background does not compete with the dot colours for attention.
`fill.colour`	The background fill colour (only exposed where missing values are present). Defaults to white. NOTE: If you change this colour, you may want to set bg.alpha to 1 to avoid the fill colour showing through
`key.top`	A number specifying the distance at top of key, defaults to 0.1
`height`	Figure height in size.units
`width`	Figure width in size.units
`size.units`	Units of size for the figure
`resolution`	Figure resolution in dpi
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`col.colour`	The colour for the column grid lines, defaults to “black”. Can be a vector.
`row.colour`	The colour for the row grid lines, defaults to “black”. Can be a vector.
`description`	Description of image/plot; default NULL.
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn

`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`border.rectangle`	Specifies the colour of the rectangle border
`lwd.rectangle`	Specifies the thickness of the rectangle border
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`dot.colour.scheme`	Colour Scheme for the dots
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`remove.symmetric`	boolean to set whether or not to remove the top left half of a symettrically sized matrix
`lwd`	line width for the axis lines

Details

It would be nice to have a library of suitable spot.size and spot.colour functions.

```
     Earlier ideas included:
   (1) Changing the dot shape to triangles, so that upward or downward-pointing dots indicated direction
   (2) Adding arrows above or below dots to indicate direction of change. This idea was not used because t
   (3) Adding line(s) in the background set at different angles to show data. This was found to be not int

A future addition may be to add the option of outlining boxes instead of adding a background. This would b
```

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`xyplot`, `levelplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);
simple.data <- data.frame(
    'A' = runif(n = 15, min = -1, max = 1),
    'B' = runif(n = 15, min = -1, max = 1),
    'C' = runif(n = 15, min = -1, max = 1),
    'D' = runif(n = 15, min = -1, max = 1),
    'E' = runif(n = 15, min = -1, max = 1)
    );

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Simple', fileext = '.tiff'),
    x = simple.data,
    main = 'Simple',
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 100
    );

# create a function to determine the spot sizes (default function works best with values < 1)
spot.size.med <- function(x) {abs(x)/3;}

# Minimal Input
create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Minimal_Input', fileext = '.tiff'),
    x = microarray[1:5,1:5],
    main = 'Minimal input',
    spot.size.function = spot.size.med,
    xaxis.rot = 90,
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 100
```

```
    );

# Axes & Labels
spot.size.small <- function(x) {abs(x)/5;}

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Axes_Labels', fileext = '.tiff'),
    x = microarray[1:15,1:15],
    main = 'Axes & labels',
    spot.size.function = spot.size.small,
    # Adjusting the font sizes and labels
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Legend
key.sizes <- seq(2,12,2);

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Legend', fileext = '.tiff'),
    x = microarray[1:15,1:15],
    main = 'Legend',
    spot.size.function = spot.size.small,
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    # Legend for dots
    key = list(
        space = 'right',
        points = list(
            cex = spot.size.small(key.sizes),
            col = default.colours(2, palette.type = 'dotmap')[2],
            pch = 19
            ),
        text = list(
            lab = as.character(key.sizes),
            cex = 1,
            adj = 1
            ),
        padding.text = 3,
        background = 'white'
        ),
```

```
    key.top = 1,
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Cluster by dots and add dendrogram
plot.data <- microarray[1:15,1:15];

# cluster data
clustered.data <- diana(plot.data);

# order data by cluster
plot.data <- plot.data[clustered.data$order,];

# create dendogram
dendrogram.data <- create.dendrogram(x = plot.data, clustering.method = 'diana',
cluster.dimension = 'row');

dendrogram.grob <- latticeExtra::dendrogramGrob(
        x = dendrogram.data,
        side = 'right',
        type = 'rectangle'
        );

# create dotmap
create.dotmap(
    x = plot.data,
    # filename = tempfile(pattern = 'Dotmap_clustered_dendrogram', fileext = '.tiff'),
    main = 'Clustered & dendrogram',
    spot.size.function = spot.size.small,
    # Adjusting the font sizes and labels
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    legend = list(
        right = list(fun = dendrogram.grob)
        ),
    right.padding = 4,
    description = 'Dotmap created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Add background data
key.sizes <- c(-1, 1);

CNA.colour.function <- function(x){
    colours <- rep('white', length(x));
    colours[sign(x) == 1] <- 'Red';
```

```
        colours[sign(x) == -1] <- 'Blue';
        colours[x == 0] <- 'transparent';
        return(colours);
        }

    create.dotmap(
        # filename = tempfile(pattern = 'Dotmap_with_Background', fileext = '.tiff'),
        # added new data for the dots
        x = CNA[1:15,1:15],
        # Moving the dot-data to be background data
        bg.data = microarray[1:15,1:15],
        colour.scheme = c('white','black'),
        main = 'Background',
        spot.size.function = 1,
        spot.colour.function = CNA.colour.function,
        xaxis.cex = 0.8,
        yaxis.cex = 0.8,
        xaxis.lab = 1:15,
        xlab.label = 'Sample',
        ylab.label = 'Gene',
        xlab.cex = 1,
        ylab.cex = 1,
        key = list(
            space = 'right',
            points = list(
                cex = 1,
                col = CNA.colour.function(key.sizes),
                pch = 19
                ),
            text = list(
                lab = c('Gain', 'Loss'),
                cex = 1,
                adj = 1
                ),
            title = 'CNA',
            padding.text = 2,
            background = 'white'
            ),
        # Adding colourkey for background data
        colourkey = TRUE,
        key.top = 1,
        description = 'Dotmap created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Discrete background colours
    create.dotmap(
        # filename = tempfile(pattern = 'Dotmap_Discrete_Background', fileext = '.tiff'),
        x = CNA[1:15,1:15],
        bg.data = microarray[1:15,1:15],
        main = 'Discrete background',
        spot.size.function = 1,
        spot.colour.function = CNA.colour.function,
```

```
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    key = list(
        space = 'right',
        points = list(
            cex = 1,
            col = CNA.colour.function(key.sizes),
            pch = 19
            ),
        text = list(
            lab = c('Gain', 'Loss'),
            cex = 1,
            adj = 1
            ),
        title = 'CNA',
        padding.text = 2,
        background = 'white'
        ),
    colourkey = TRUE,
    key.top = 1,
    # Changing background colour scheme
    colour.scheme = c('lightyellow','gold','darkorange', 'darkorange3'),
    at = seq(0,12,3),
    colourkey.labels = seq(0,12,3),
    colourkey.labels.at = seq(0,12,3),
    bg.alpha = 0.65,
    description = 'Dotmap created by BoutrosLab.plotting.general'
    );

# Dot outlines
border.colours <- function(x){
    colours <- rep('transparent', length(x));
    colours[x > 0] <- 'black';
    colours[x == 0] <- 'transparent';
    return(colours);
    }

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Outlined_Dots', fileext = '.tiff'),
    x = CNA[1:15,1:15],
    bg.data = microarray[1:15,1:15],
    main = 'Dot outlines',
    spot.size.function = 1,
    spot.colour.function = CNA.colour.function,
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
```

```
        ylab.label = 'Gene',
        xlab.cex = 1,
        ylab.cex = 1,
        key = list(
            space = 'right',
            points = list(
                cex = 1,
                col = 'black',
                # Remember to also change the pch in the legend
                pch = 21,
                fill = CNA.colour.function(key.sizes)
                ),
            text = list(
                lab = c('Gain', 'Loss'),
                cex = 1,
                adj = 1
                ),
            title = 'CNA',
            padding.text = 2,
            background = 'white'
            ),
        colourkey = TRUE,
        key.top = 1,
        colour.scheme = c('lightyellow','gold','darkorange', 'darkorange3'),
        at = seq(0,12,3),
        colourkey.labels = seq(0,12,3),
        colourkey.labels.at = seq(0,12,3),
        bg.alpha = 0.65,
        # Change the plotting character to one which has an outline
        pch = 21,
        pch.border.col = border.colours(CNA[1:15,1:15]),
        description = 'Dotmap created by BoutrosLab.plotting.general'
        );

    # Covariates & Legend
    sex.colours <- patient$sex[1:15];
    sex.colours[sex.colours == 'male'] <- 'dodgerblue';
    sex.colours[sex.colours == 'female'] <- 'pink';

    sample.covariate <- list(
        rect = list(
            col = 'black',
            fill = sex.colours,
            lwd = 1.5
            )
        );

    cov.grob <- covariates.grob(
        covariates = sample.covariate,
        ord = c(1:15),
        side = 'top'
        );
```

```
sample.cov.legend <- list(
    legend = list(
        colours = c('dodgerblue', 'pink'),
        labels = c('male','female'),
        title = 'Sex'
        )
    );

cov.legend <- legend.grob(
    legends = sample.cov.legend
    );

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Covariates', fileext = '.tiff'),
    x = CNA[1:15,1:15],
    bg.data = microarray[1:15,1:15],
    main = 'Covariates',
    spot.size.function = 1,
    spot.colour.function = CNA.colour.function,
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    key = list(
        space = 'right',
        points = list(
            cex = 1,
            col = 'black',
            pch = 21,
            fill = CNA.colour.function(key.sizes)
            ),
        text = list(
            lab = c('Gain', 'Loss'),
            cex = 1,
            adj = 1
            ),
        title = 'CNA',
        padding.text = 2,
        background = 'white'
        ),
    colourkey = TRUE,
    key.top = 1,
    colour.scheme = c('lightyellow','gold','darkorange', 'darkorange3'),
    at = seq(0,12,3),
    colourkey.labels = seq(0,12,3),
    colourkey.labels.at = seq(0,12,3),
    bg.alpha = 0.65,
    pch = 21,
    pch.border.col = border.colours(CNA[1:15,1:15]),
    # Insert covariates & legend
```

```
        legend = list(
            top = list(
                fun = cov.grob
                ),
            left = list(
                fun = cov.legend
                )
            ),
        description = 'Dotmap created by BoutrosLab.plotting.general'
        );

    # Side covariates with label
    chr.cov.colours <- microarray$Chr;
    chr.cov.colours[microarray$Chr == 1] <- default.colours(3, palette.type = 'chromosomes')[1];
    chr.cov.colours[microarray$Chr == 2] <- default.colours(3, palette.type = 'chromosomes')[2];
    chr.cov.colours[microarray$Chr == 3] <- default.colours(3, palette.type = 'chromosomes')[3];

    chr.covariate <- list(
        rect = list(
            col = 'white',
            fill = chr.cov.colours,
            lwd = 1.5
            )
        );

    chr.cov.grob <- covariates.grob(
        covariates = chr.covariate,
        ord = c(1:15),
        side = 'right'
        );

    # create dot legend
    dot.grob <- draw.key(
        list(
            space = 'right',
             points = list(
                    cex = 1,
                    col = 'black',
                    pch = 21,
                    fill = CNA.colour.function(key.sizes)
                    ),
                text = list(
                    lab = c('Gain', 'Loss'),
                    cex = 1,
                    adj = 1
                    ),
                title = 'CNA',
                padding.text = 2,
                background = 'white'
            )
        );

    # Setting up the layout for the joint legends
```

```
right.layout <- grid.layout(
    nrow = 1,
    ncol = 2,
    width = unit(
        x = c(0,1),
        units = rep('lines',2)
        ),
    heights = unit(
        x = c(1,1),
        units = rep('npc', 1)
        )
    );

right.grob <- frameGrob(layout = right.layout);

right.grob <- packGrob(
    frame = right.grob,
    grob = chr.cov.grob,
    row = 1,
    col = 1
    );

right.grob <- packGrob(
    frame = right.grob,
    grob = dot.grob,
    row = 1,
    col = 2
    );

temp <- create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_Covariates_Side', fileext = '.tiff'),
    x = CNA[1:15,1:15],
    bg.data = microarray[1:15,1:15],
    main = 'Both covariates',
    spot.size.function = 1,
    spot.colour.function = CNA.colour.function,
    xaxis.cex = 0.8,
    yaxis.cex = 0.8,
    xaxis.lab = 1:15,
    xlab.label = 'Sample',
    ylab.label = 'Gene',
    xlab.cex = 1,
    ylab.cex = 1,
    colourkey = TRUE,
    key.top = 1,
    colour.scheme = c('lightyellow','gold','darkorange', 'darkorange3'),
    at = seq(0,12,3),
    colourkey.labels = seq(0,12,3),
    colourkey.labels.at = seq(0,12,3),
    bg.alpha = 0.65,
    pch = 21,
    pch.border.col = border.colours(CNA[1:15,1:15]),
    # insert covariates & legend
```

```
        legend = list(
            right = list(
                fun = right.grob
                )
            ),
        description = 'Dotmap created by BoutrosLab.plotting.general'
        );

    # add side label to covariate
    print(temp, position = c(0,0,1,1), more = TRUE);

    draw.key(
        key = list(
            text = list(
                lab = 'Covariate Label',
                cex = 1,
                adj = 1
                )
            ),
        # position label on the plot
        vp = viewport(x = 0.86, y = 0.155, height = 1, width = 0.5, angle = 90),
        draw = TRUE
        );

    dev.off();

    # Nature style
    create.dotmap(
        # filename = tempfile(pattern = 'Dotmap_Nature_style', fileext = '.tiff'),
        x = CNA[1:15,1:15],
        bg.data = microarray[1:15,1:15],
        main = 'Nature style',
        spot.size.function = 1,
        spot.colour.function = CNA.colour.function,
        xaxis.cex = 0.8,
        yaxis.cex = 0.8,
        xaxis.lab = 1:15,
        xlab.cex = 1,
        ylab.cex = 1,
        key = list(
            space = 'right',
            points = list(
                cex = 1,
                col = 'black',
                # Remember to also change the pch in the legend
                pch = 21,
                fill = CNA.colour.function(key.sizes)
                ),
            text = list(
                lab = c('Gain', 'Loss'),
                cex = 1,
                adj = 1
                ),
```

```
        title = 'CNA',
        padding.text = 2,
        background = 'white'
        ),
    colourkey = TRUE,
    key.top = 1,
    colour.scheme = c('lightyellow','gold','darkorange', 'darkorange3'),
    at = seq(0,12,3),
    colourkey.labels = seq(0,12,3),
    colourkey.labels.at = seq(0,12,3),
    bg.alpha = 0.65,
    # Change the plotting character to one which has an outline
    pch = 21,
    pch.border.col = border.colours(CNA[1:15,1:15]),

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    resolution = 200
    );

simple.data.sym <- data.frame(
    '1' = runif(n = 7, min = -1, max = 1),
    '2' = runif(n = 7, min = -1, max = 1),
    '3' = runif(n = 7, min = -1, max = 1),
    '4' = runif(n = 7, min = -1, max = 1),
    '5' = runif(n = 7, min = -1, max = 1),
    '6' = runif(n = 7, min = -1, max = 1),
    '7' = runif(n = 7, min = -1, max = 1)
    );

create.dotmap(
    # filename = tempfile(pattern = 'Dotmap_remove_symmetric', fileext = '.tiff'),
    x = simple.data.sym,
    main = 'Simple',
    xaxis.lab = seq(1,7,1),
    description = 'Dotmap created by BoutrosLab.plotting.general',
    remove.symmetric = TRUE,
    resolution = 200
    );
```

`create.gif` *Make a gif*

Description

Takes a function and several sets of parameters and makes a gif of their function calls

Usage

```
create.gif(
exec.func,
parameters,
number.of.frames,
delay = 40,
filename)
```

Arguments

`exec.func`	The function that will be used to make the plots for the gif
`parameters`	Parameter list to be sent to the exec func at each frame
`number.of.frames`	Total number of frames to be made (must match number of parameter lists)
`delay`	Delay between each frame in the gif
`filename`	Name of output file (must end in .gif)

Author(s)

Jeffrey Green

See Also

`stripplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(223);

simple.data1 <- data.frame(
    x = sample(1:15, 10),
    y = LETTERS[1:10]
    );

simple.data2 <- data.frame(
    x = sample(1:15, 10),
    y = LETTERS[1:10]
    );

simple.data3 <- data.frame(
    x = sample(1:15, 10),
    y = LETTERS[1:10]
    );

p = list(
```

```
list(formula = x ~ y,data = simple.data1, yat = seq(0,16,2)),
list(formula = x ~ y,data = simple.data2, yat = seq(0,16,2)),
list(formula = x ~ y,data = simple.data3, yat = seq(0,16,2))
)

create.gif(
exec.func = create.barplot,
parameters = p,
number.of.frames = 3,
delay = 20,
filename = tempfile(pattern = 'test', fileext = '.gif')
)
```

`create.heatmap` *Make a heatmap*

Description

Takes a data.frame and creates a heatmap

Usage

```
create.heatmap(
x,
filename = NULL,
clustering.method = 'diana',
cluster.dimensions = 'both',
rows.distance.method = 'correlation',
cols.distance.method = 'correlation',
cor.method = 'pearson',
row.dendrogram = list(),
col.dendrogram = list(),
plot.dendrograms = 'both',
force.clustering = FALSE,
criteria.list = TRUE,
covariates = list(),
covariates.grid.row = NULL,
covariates.grid.col = NULL,
covariates.grid.border = NULL,
covariates.row.lines = NULL,
covariates.col.lines = NULL,
covariates.reorder.grid.index = FALSE,
covariates.padding = 0.25,
covariates.top = list(),
covariates.top.grid.row = NULL,
covariates.top.grid.col = NULL,
covariates.top.grid.border = NULL,
```

```
covariates.top.row.lines = NULL,
covariates.top.col.lines = NULL,
covariates.top.reorder.grid.index = FALSE,
covariates.top.padding = 0.25,
covariate.legends = list(),
legend.cex = 1,
legend.title.cex = 1,
legend.title.just = 'centre',
legend.title.fontface = 'bold',
legend.border = NULL,
legend.border.padding = 1,
legend.layout = NULL,
legend.between.col = 1,
legend.between.row = 1,
legend.side = 'left',
main = list(label = ''),
main.just = "center",
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
right.size.add = 1,
top.size.add = 1,
right.dendrogram.size = 2.5,
top.dendrogram.size = 2.5,
scale.data = FALSE,
yaxis.lab = NULL,
xaxis.lab = NULL,
xaxis.lab.top = NULL,
xaxis.cex = 1.5,
xaxis.top.cex = NULL,
yaxis.cex = 1.5,
xlab.cex = 2,
ylab.cex = 2,
xlab.top.label = NULL,
        xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = "center",
xlab.top.x = 0.5,
xlab.top.y = 0,
xat = TRUE,
xat.top = NULL,
yat = TRUE,
xaxis.tck = NULL,
xaxis.top.tck = NULL,
yaxis.tck = NULL,
xaxis.col = 'black',
yaxis.col = 'black',
col.pos = NULL,
```

```
row.pos = NULL,
cell.text = '',
text.fontface = 1,
text.cex = 1,
text.col = 'black',
text.position = NULL,
text.offset = 0,
text.use.grid.coordinates = TRUE,
colourkey.cex = 3.6,
xaxis.rot = 90,
xaxis.rot.top = 90,
yaxis.rot = 0,
xlab.label = '' ,
ylab.label = '',
xlab.col = 'black',
ylab.col = 'black',
axes.lwd = 2,
gridline.order = 'h',
grid.row = FALSE,
grid.col = FALSE,
force.grid.row = FALSE,
force.grid.col = FALSE,
grid.limit = 50,
row.lines = seq(0, ncol(x), 1) + 0.5,
col.lines = seq(0, nrow(x), 1) + 0.5,
colour.scheme = c(),
total.colours = 99,
colour.centering.value = 0,
colour.alpha = 1,
fill.colour = 'darkgray',
at = NULL,
print.colour.key = TRUE,
colourkey.labels.at = NULL,
colourkey.labels = NULL,
top.padding = 0.1,
bottom.padding = 0.5,
right.padding = 0.5,
left.padding = 0.5,
x.alternating = 1,
shrink = 1,
row.colour = 'black',
col.colour = 'black',
row.lwd = 1,
col.lwd = 1,
grid.colour = NULL,
grid.lwd = NULL,
width = 6,
height = 6,
```

```
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
xaxis.covariates = NULL,
xaxis.covariates.y = 0,
yaxis.covariates = NULL,
yaxis.covariates.x = NULL,
description = 'Created with BoutrosLab.plotting.general',
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
symbols = list(borders = NULL,
squares = NULL,
circles = NULL),
same.as.matrix = FALSE,
input.colours = FALSE,
axis.xlab.padding = 0.1,
stratified.clusters.rows = NULL,
stratified.clusters.cols = NULL,
        inside.legend = NULL,
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE
);
```

Arguments

`x`	Either a data-frame or a matrix from which the heatmap is to created
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`clustering.method`	Method used to cluster the records – “none” gives unclustered data. Accepts all agglomerative clustering methods available in hclust, plus “diana” (which is divisive).
`cluster.dimensions`	Should clustering be performed on rows, columns, or both – supersedes setting of plot.dendrograms
`rows.distance.method`	Method name of the distance measure between rows to be used for clustering. Defaults to “correlation”. Other supported methods are same as in ?dist. Also supports “jaccard” which is useful for clustering categorical variables. “euclidean” is sometimes more robust when ties cause “Unclusterable matrix: some col-distances are null” errors. Note, rows and cols are switched due an internal transposition of the data.
`cols.distance.method`	Method name of the distance measure between columns to be used for clustering. Defaults to “correlation”. Other supported methods are same as in ?dist. Also supports “jaccard” which is useful for clustering categorical variables. “euclidean” is sometimes more robust when ties cause “Unclusterable matrix: some

	col-distances are null" errors. Note, rows and cols are switched due an internal transposition of the data.
`cor.method`	The method used for calculating correlation. Defaults to "pearson"
`row.dendrogram`	A dendrogram object corresponding to the heatmap rows. If provided, row clustering cannot be performed
`col.dendrogram`	A dendrogram object corresponding to the heatmap columns. If provided, column clustering cannot be performed
`plot.dendrograms`	If clustering is performed or dendrograms are provided, which dendrograms should be plotted – "none", "right", "top", or "both"
`force.clustering`	Binary to over-ride the control that prevents clustering of too-large matrices
`criteria.list`	A vector indicating which rows should be retained
`covariates`	Any row-wise covariate annotate to add to the plot, as a fully formed list (placed on right side of plot)
`covariates.grid.row`	A list of parameters passed to `gpar` specifying the behaviour of row lines in the right covariate bars
`covariates.grid.col`	A list of parameters passed to `gpar` specifying the behaviour of column lines in the right covariate bars
`covariates.grid.border`	A list of parameters passed to `gpar` specifying the behaviour of the border around the right covariate bars
`covariates.row.lines`	Vector of row indices where grid lines should be drawn on the right covariate bars. If NULL (default), all row lines are drawn. Ignored if `covariates.grid.row` is not specified
`covariates.col.lines`	Vector of column indices where grid lines should be drawn on the right covariate bars. If NULL (default), all column lines are drawn. Ignored if `covariates.grid.col` is not specified
`covariates.reorder.grid.index`	Boolean specifying whether grid line indices for the right covariate bars should be re-ordered with clustering
`covariates.padding`	Amount of empty space (in "lines") to place between the right covariate bars and dendrogram
`covariates.top`	Any column-wise covariate annotate to add to the plot, as a fully formed list
`covariates.top.grid.row`	A list of parameters passed to `gpar` specifying the behaviour of row lines in the top covariate bars
`covariates.top.grid.col`	A list of parameters passed to `gpar` specifying the behaviour of column lines in the top covariate bars

`covariates.top.grid.border`	A list of parameters passed to `gpar` specifying the behaviour of the border around the top covariate bars
`covariates.top.row.lines`	Vector of row indices where grid lines should be drawn on the top covariate bars. If NULL (default), all row lines are drawn. Ignored if `covariates.top.grid.row` is not specified
`covariates.top.col.lines`	Vector of column indices where grid lines should be drawn on the top covariate bars. If NULL (default), all column lines are drawn. Ignored if `covariates.top.grid.col` is not specified
`covariates.top.reorder.grid.index`	Boolean specifying whether grid line indices for the top covariate bars should be re-ordered with clustering
`covariates.top.padding`	Amount of empty space (in "lines") to place between the top covariate bars and dendrogram
`covariate.legends`	A list defining covariate legends to add to the plot. See `legends` argument of `legend.grob` for more information
`legend.cex`	Size of text labels in covariate legends, defaults to 1
`legend.title.cex`	Size of title text in covariate legends, defaults to 1
`legend.title.just`	Justification of title text in covariate legends, defaults to "centre"
`legend.title.fontface`	Font face of title text in covariate legends – "plain", "bold", "italic", etc.
`legend.border`	A list of parameters passed to `gpar` specifying line options for the legend border, defaults to NULL (no border drawn)
`legend.border.padding`	The amount of empty space (split equally on both sides) to add between the legend and its border, in "lines" units
`legend.layout`	Numeric vector of length 2 specifying the number of columns and rows for the legend layout, defaults to a logical layout based on `legend.side`
`legend.between.col`	Amount of space to add between columns in the layout, in "lines" units
`legend.between.row`	Amount of space to add between rows in the layout, in "lines" units
`legend.side`	Side of the plot where the legends should be drawn – "left", "right", or "top"
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 2.5

`right.size.add`	The size of each extra covariate row in the right dendrogram in units of “lines”
`top.size.add`	The size of each extra covariate row in the top dendrogram in units of “lines”
`right.dendrogram.size`	Size of right dendrogram
`top.dendrogram.size`	Size of top dendrogram
`scale.data`	TRUE/FALSE to do row-wise scaling with mean-centering and sd-scaling
`xaxis.lab`	A vector of row labels, NA = use existing rownames, NULL = none
`xaxis.lab.top`	The label for the top x-axis. Required only if you want to print a top *and* bottom xaxis, otherwise use x.alternating = 2 for top axis only. Defaults to NULL
`yaxis.lab`	A vector of col labels, NA = use existing colnames, NULL = none
`xaxis.cex`	Size of x-axis label text - defaults to values found in a look-up table
`xaxis.top.cex`	Size of top x axis label text
`yaxis.cex`	Size of y-axis label text - defaults to values found in a look-up table
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 90
`xaxis.rot.top`	Rotation of the top x-axis tick labels; defaults to 90
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`xat.top`	Vector listing where the x-axis labels should be drawn on the top of the plot. Required only when you want bottom and top axis, otherwise use x.alternating = 2, to get top axis only. Defaults to NULL
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`xaxis.tck`	Size of x-axis tick marks. Defaults to NULL for intelligent choice based on covariate size.

`xaxis.top.tck`	Size of top x-axis tick marks. Defaults to NULL for intelligent choice based on covariate size.
`yaxis.tck`	Size of y-axis tick marks. Defaults to NULL for intelligent choice based on covariate size.
`col.pos`	Vector of column positions for adding text to cell, defaults to NULL
`row.pos`	Vector of row positions for adding text to cell, defaults to NULL
`cell.text`	Text to add to cell, defaults to an empty string
`text.fontface`	1 = Plain, 2 = Bold, 3 = Italic, default is 1
`text.cex`	Text size, default is 1
`text.col`	Text colour, default is black.
`text.position`	The position of the text, defaults to center.
`text.offset`	The offset of the position, defaults to 0.
`text.use.grid.coordinates`	Indetifier if grid coordinates or npc coordinates should be used
`colourkey.cex`	Size of colourkey label text
`axes.lwd`	Width of heatmap border. Note it also changes the colourkey border and ticks
`gridline.order`	Character specifying order in which to draw interior grid-lines ('h' or 'v'). Defaults to 'h' for horizontal first.
`grid.row`	Allow turning off of the interior grid-lines. Default FALSE
`grid.col`	Allow turning off of the interior grid-lines. Default FALSE
`force.grid.row`	Overrides default behaviour of turning off grid lines when number of rows exceed grid.limit. Defaults to FALSE
`force.grid.col`	Overrides default behaviour of turning off grid lines when number of columns exceed grid.limit. Defaults to FALSE
`grid.limit`	Limit set for when to turn off column and row lines if data size exceeds it. Defaults to 50
`row.lines`	Vector specifying location of lines, default is seq(1, ncol(x), 1) + 0.5. Note: Add 0.5 to customized vector
`col.lines`	Vector specifying location of lines, default is seq(1, nrow(x), 1) + 0.5. Note: Add 0.5 to customized vector
`colour.scheme`	Heatmap colouring. Accepts old-style themes, or a vector of either two or three colours that are gradiated to create the final palette.
`total.colours`	Total number of colours to plot
`colour.centering.value`	What should be the center of the colour-map
`colour.alpha`	Bias to be added to colour selection (uses x^colour.alpha in mapping). Set to “automatic” for auto-adjustment.
`fill.colour`	The background fill (only exposed where missing values are present
`print.colour.key`	Should the colour key be printed at all?

`at`	A vector specifying the breakpoints along the range of x; each interval specified by these breakpoints are assigned to a colour from the palette. Defaults to NULL, which corresponds to the range of x being divided into total.colours equally spaced intervals. If x has values outside of the range specified by “at” those values are shown with the colours corresponding to the extreme ends of the colour spectrum and a warning is given.
`colourkey.labels.at`	A vector specifying the tick-positions on the colourkey
`colourkey.labels`	A vector specifying tick-labels of the colourkey
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.5
`right.padding`	A number specifying the distance to the right margin, defaults to 0.5
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`x.alternating`	A value specifying the position of the col names, defaults to 1. 1 means below the graph, 2 means above the graph. Use 3 to get tick marks below and above graph, but still need to specify xat.top and xaxis.lab.top to get values there
`shrink`	Allows rectangles to be scaled, defaults to 1
`row.colour`	Interior grid-line colour, defaults to “black”. Can be a vector
`col.colour`	Interior grid-line colour, defaults to “black”. Can be a vector
`row.lwd`	Interior grid-line width, defaults to 1. Setting to zero is equivalent to grid.row = FALSE and grid.col = FALSE. Can be a vector.
`col.lwd`	Interior grid-line width, defaults to 1. Setting to zero is equivalent to grid.row = FALSE and grid.col = FALSE. Can be a vector.
`grid.colour`	Interior grid-line colour, defaults to “black”. Can be a vector. Applies to both rows and columns. DEPRECATED
`grid.lwd`	Interior grid-line width, defaults to 1. Setting to zero is equivalent to grid.row = FALSE and grid.col = FALSE. Applies to both rows and columns. DEPRECATED
`width`	Figure width in size.units
`height`	Figure height in size.units
`size.units`	Units of size for the figure
`resolution`	Figure resolution in dpi
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`xaxis.covariates`	Any column-wise covariate annotate to add to the plot, as a fully formed list
`xaxis.covariates.y`	The y coordinate of the location of the x axis covariates
`yaxis.covariates`	Any row-wise covariate annotate to add to the plot, as a fully formed list
`yaxis.covariates.x`	The x coordinate of the lcoation of the y axis covariates

`description`	Short description of image/plot; default NULL.
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`symbols`	Extra symbols to be added (borders, squares and circles)
`same.as.matrix`	Prevents the flipping of the matrix that the function normally does
`input.colours`	boolean expressing whether or not the matrix was specified using colours or integer values. Defaults to FALSE
`axis.xlab.padding`	Padding between axis of plot and x label
`stratified.clusters.rows`	the row locations of the rows to be combined into a strata
`stratified.clusters.cols`	the column locations of the columns to be combined into a strata
`inside.legend`	legend specification for the inside legend/key of the heatmap
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the heatmap. In particular, if a script that uses such a call of create heatmap is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Note that we would very much like to be able to pass `xaxis.cex` and `yaxis.cex` as vectors of the same length as the actual data-table. However lattice does not support that, because it currently expects them as a two-element vectors to specify left/right or top/bottom axes separately. I’ve raised a bug report on requesting an enhancement, but this would require an API change so... not sure if it will happen. Here’s the bug-report: https://r-forge.r-project.org/tracker/index.php?func=detail&aid=1702&group_id=638&a

Author(s)

Paul C. Boutros

See Also

`covariates.grob`, `create.dendrogram`, `legend.grob`

Examples

```
set.seed(12345);
simple.data <- data.frame(
    x <- rnorm(n = 15),
    y <- rnorm(n = 15),
    z <- rnorm(n = 15),
    v <- rnorm(n = 15),
    w <- rnorm(n = 15)
    );

simple.1D.data <- data.frame(x = rnorm(n = 15));


create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_1D_Inside_Legend', fileext = '.tiff'),
    x = simple.1D.data,
    clustering.method='none',
    inside.legend = list(fun = draw.key,
        args = list(
    key = list(
                text = list(
                    lab = c('test','test','test','test'),
                    cex = 1,
                    fontface = 'bold'
                    ),
                padding.text = 3,
                background = 'white',
alpha.background = 0
                )
            ),
        x = 0.5,
        y = 0.5
        ),
    resolution = 100
    )

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Simple', fileext = '.tiff'),
    x = simple.data,
    main = 'Simple',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 100
    );
```

```
simple.data.col <- data.frame(
    x <- c('blue','green','red','yellow','blue','red','black','white','purple','grey'),
    y <- rep('red',10),
    z <- rep('yellow',10),
    v <- rep('green',10),
    w <- rep('purple',10)
    );

# Input Colours Provided
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Simple_Using_Colours', fileext = '.tiff'),
    x = simple.data.col,
    clustering.method = 'none',
    input.colours = TRUE,
    resolution = 100
    );

# Single Input Colour Provided
create.heatmap(
   # filename = tempfile(pattern = 'Heatmap_Simple_Using_Single_Colour', fileext = '.tiff'),
    x = simple.data.col[, ncol(simple.data.col), drop = FALSE],
    clustering.method = 'none',
    input.colours = TRUE,
    resolution = 100
    );

# Minimal Input
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Minimal_Input', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Minimal input',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes and labels
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Axes_Labels', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Axes & labels',
    # Changing axes
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    # Turning on default row and column labels
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    # Adjusting font sizes
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    # Changing colourkey
    colourkey.cex = 1,
```

```
    colourkey.labels.at = seq(2,12,1),
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 100
    );


# Custom Axes
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Custom_Axes', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Customized axes',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    # Specify where to place tick marks
    colourkey.labels.at = c(3,4, 6, 7, 10, 11),
    # Specify label colours (note: this is based on the pre-clustering order)
   xaxis.col = c('black', 'red',rep('black',6), 'red','black', 'black','red',rep('black',8)),
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Two-sided Colour Scheme
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Colour_Scheme_1', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Colour scheme',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    # Changing the colours
    colour.scheme = c('white','firebrick'),
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Three-sided Colour Scheme
# Note: when using a three-sided colour scheme, it is advised to have two-sided data
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Colour_Scheme_2', fileext = '.tiff'),
```

```
    x = microarray[1:20, 1:20],
    main = 'Colour scheme',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    # Changing the colours
    colour.scheme = c('red','white','turquoise'),
    # Scale the data to center around the mean
    scale.data = TRUE,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Colour Alpha
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Colour_Alpha', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Colours alpha',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    # Adjusting the alpha value of the colours
    colour.alpha = 'automatic',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Clustering
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_No_Clustering', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'No clustering',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
```

```
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    # Turning clustering off
    clustering.method = 'none',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Clustering
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Clustering_Methods', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Clustering methods',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    # Clustering method defaults to 'diana', but can be set to other options
    clustering.method = 'complete',
    # Also setting the distance measures
    rows.distance.method = 'euclidean',
    cols.distance.method = 'manhattan',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Stratified Clustering
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Stratified_Clustering', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Stratified clustering',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    # Stratifying the clustering by rows
    stratified.clusters.rows = list(c(1:10), c(11:20)),
    # Adding line to show highlight the division between the two strata
```

```
        grid.row = TRUE,
        row.lines = 10.5,
        row.lwd = 2,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
        );

    # Dendrogram provided
    col.dendrogram <- BoutrosLab.plotting.general::create.dendrogram(
        x = microarray[1:20, 1:20],
        cluster.dimension = 'col'
        );

    row.dendrogram <- BoutrosLab.plotting.general::create.dendrogram(
        x = microarray[1:20, 1:20],
        cluster.dimension = 'row'
        );

    create.heatmap(
        # filename = tempfile(pattern = 'Heatmap_Dendrogram_Provided', fileext = '.tiff'),
        x = microarray[1:20, 1:20],
        main = 'Dendrogram provided',
        xlab.label = 'Genes',
        ylab.label = 'Samples',
        xaxis.lab = NA,
        yaxis.lab = 1:20,
        xaxis.cex = 0.75,
        yaxis.cex = 0.75,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        colourkey.cex = 1,
        colourkey.labels.at = seq(2,12,1),
        colour.alpha = 'automatic',
       # note: row/column dendrograms are switched because the function inverts rows and columns
        clustering.method = 'none',
        row.dendrogram = col.dendrogram,
        col.dendrogram = row.dendrogram,
        # Adjusting the size of the dendrogram
        right.dendrogram.size = 3,
        top.dendrogram.size = 2.5,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
        );

    # Covariates and Legends
    # Note: covariates can also be added using the create.multiplot function
    # set the colour schemes for the covariates
    sex.colours <- patient$sex;
    sex.colours[sex.colours == 'male'] <- 'dodgerblue';
    sex.colours[sex.colours == 'female'] <- 'pink';

    stage.colours <- patient$stage;
    stage.colours[stage.colours == 'I'] <- 'plum1';
```

```
stage.colours[stage.colours == 'II'] <- 'orchid1';
stage.colours[stage.colours == 'III'] <- 'orchid3';
stage.colours[stage.colours == 'IV'] <- 'orchid4';

# create an object to draw the covariates from
sample.covariate <- list(
    rect = list(
        col = 'black',
        fill = sex.colours,
        lwd = 1.5
        ),
    rect = list(
        col = 'black',
        fill = stage.colours,
        lwd = 1.5
        )
    );

# create a legend for the covariates
sample.cov.legend <- list(
    legend = list(
        colours = c('dodgerblue', 'pink'),
        labels = c('male','female'),
        title = 'Sex'
        ),
    legend = list(
        colours = c('plum1', 'orchid1','orchid3', 'orchid4'),
        labels = c('I','II', 'III', 'IV'),
        title = 'Stage'
        )
    );

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Covariates_Simple', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Covariates',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    # adding covariates and corresponding legend
    covariates = sample.covariate,
    covariate.legend = sample.cov.legend,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# Top covariate and legend customization
chr.cov.colours <- microarray$Chr;
chr.cov.colours[microarray$Chr == 1] <- default.colours(3, palette.type = 'chromosomes')[1];
chr.cov.colours[microarray$Chr == 2] <- default.colours(3, palette.type = 'chromosomes')[2];
chr.cov.colours[microarray$Chr == 3] <- default.colours(3, palette.type = 'chromosomes')[3];

chr.covariate <- list(
    rect = list(
        col = 'white',
        fill = chr.cov.colours,
        lwd = 1.5
        )
    );

# join covariate legends
combo.cov.legend <- list(
    legend = list(
        colours = default.colours(3, palette.type = 'chromosomes'),
        labels = c('1','2', '3'),
        title = 'Chromosome',
        border = 'white'
        ),
    legend = list(
        colours = c('dodgerblue', 'pink'),
        labels = c('male','female'),
        title = 'Sex'
        ),
    legend = list(
        colours = c('plum1', 'orchid1','orchid3', 'orchid4'),
        labels = c('I','II', 'III', 'IV'),
        title = 'Stage'
        )
    );

create.heatmap(
   # filename = tempfile(pattern = 'Heatmap_Covariate_Legend_Custom', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Custom covariates & legend',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    clustering.method = 'none',
    # side covariate
    covariates = sample.covariate,
```

```
    # top covariate and covariate border specification
    covariates.top = chr.covariate,
    covariate.legend = combo.cov.legend,
    # making outline of border a matching green
    covariates.top.grid.border = list(col = 'lightblue', lwd = 2),
    # making certain column divisions a different colour
    covariates.top.col.lines = c(5,6),
    covariates.top.grid.col = list(col = 'blue', lwd = 2),
    # legend customization
    legend.side = c('right','left','top'),
    legend.title.cex = 0.75,
    legend.cex = 0.75,
    legend.title.just = 'left',
    legend.border = list(lwd = 1),
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Custom gridlines

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Gridlines', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Gridlines',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    # colouring gridlines
    grid.row = TRUE,
    grid.col = TRUE,
    row.colour = 'white',
    col.colour = 'white',
    row.lwd = 1.5,
    col.lwd = 1.5,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Label cells
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Labelled_Cells', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Labelled cells',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
```

```
        xaxis.cex = 0.75,
        yaxis.cex = 0.75,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        colourkey.cex = 1,
        colourkey.labels.at = seq(2,12,1),
        colour.alpha = 'automatic',
        grid.row = TRUE,
        grid.col = TRUE,
        row.colour = 'white',
        col.colour = 'white',
        row.lwd = 1.5,
        col.lwd = 1.5,
        clustering.method = 'none',
        # conditionally labelling cells
        # flipping rows and columns because the heatmap function does so
        row.pos = which(microarray[1:20, 1:20] > 11, arr.ind = TRUE)[,2],
        col.pos = which(microarray[1:20, 1:20] > 11, arr.ind = TRUE)[,1],
        cell.text = microarray[1:20, 1:20][microarray[1:20, 1:20] > 11],
        text.col = 'white',
        text.cex = 0.65,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
        );
    # Label cells
    create.heatmap(
        # filename = tempfile(pattern = 'Heatmap_Labelled_Cells_NPC', fileext = '.tiff'),
        x = microarray[1:20, 1:20],
        main = 'Labelled cells',
        xlab.label = 'Genes',
        ylab.label = 'Samples',
        xaxis.cex = 0.75,
        yaxis.cex = 0.75,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        colourkey.cex = 1,
        colourkey.labels.at = seq(2,12,1),
        colour.alpha = 'automatic',
        grid.row = TRUE,
        grid.col = TRUE,
        row.colour = 'white',
        col.colour = 'white',
        row.lwd = 1.5,
        col.lwd = 1.5,
        clustering.method = 'none',
        text.use.grid.coordinates = FALSE,
        # conditionally labelling cells
        # flipping rows and columns because the heatmap function does so
        cell.text = c("text1","text2"),
        text.col = 'white',
        text.cex = 0.65,
        text.position = list(c(0.5,0.5),c(0.75,0.75)),
        description = 'Heatmap created using BoutrosLab.plotting.general',
```

```
    resolution = 200
    );

# Method 1 of adding symbols (very similar to how text is added)
points <- microarray[1:20, 1:20][microarray[1:20, 1:20] > 11];
size.from <- range(points, na.rm = TRUE);
size.to <- c(1,3);
point.size <- (points - size.from[1])/diff(size.from) * diff(size.to) + size.to[1];
point.colour <- grey(runif(sum(microarray[1:20, 1:20] > 11), max = 0.5));

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Symbols_1', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Symbols',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    clustering.method = 'none',
    # conditionally adding points to cells
    # flipping rows and columns because the heatmap function does so
    row.pos = which(microarray[1:20, 1:20] > 11, arr.ind = TRUE)[,2],
    col.pos = which(microarray[1:20, 1:20] > 11, arr.ind = TRUE)[,1],
    cell.text = rep(expression(”\u25CF”), times = sum(microarray[1:20, 1:20] > 11)),
    text.col = point.colour,
    text.cex = point.size,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Method 2 of Adding Symbols
# Create matrices to describe the symbols
circle.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = FALSE
    );

circle.colour.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 'pink'
    );

circle.size.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 20
```

```
    );

border.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = FALSE
    );

border.colour.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 'black'
    );

border.size.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 4
    );

square.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = FALSE
    );

square.colour.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 'pink'
    );

square.size.matrix <- matrix(
    nrow = 20,
    ncol = 20,
    data = 10
    );

# setting up the symbols
symbol.locations <- list(
    circles = list(
        list(
            x = circle.matrix,
            col = circle.colour.matrix,
            size = circle.size.matrix
            )
        ),
    borders = list(
        list(
            x = border.matrix,
            col = border.colour.matrix,
            size = border.size.matrix
            ),
```

```
        # creating a border encompassing a larger area
        list(
            xright = 12.10,
            xleft = 12,
            ybottom = 1,
            ytop = 20,
            size = 4,
            col = 'pink'
            )
        ),
    squares = list(
        list(
            x = square.matrix,
            col = square.colour.matrix,
            size = square.size.matrix
            )
        )
    );

# Set which items in the matrix will be shown

# symbol.locations$borders[[1]]$x <- FALSE;
# symbol.locations$squares[[1]]$x <- FALSE;
symbol.locations$circles[[1]]$x[which(microarray[1:20,1:20] > 11, arr.ind = TRUE)] <- TRUE;

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Symbols_2', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Symbols',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    colourkey.labels.at = seq(2,12,1),
    colour.alpha = 'automatic',
    clustering.method = 'none',
    # adding symbols
    symbols = symbol.locations,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Rotate matrix
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Rotated_Matrix', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Rotated matrix',
    # Also flip labels
    ylab.label = 'Genes',
    xlab.label = 'Samples',
```

```
        xaxis.lab = NA,
        yaxis.lab = NA,
        xaxis.cex = 0.75,
        yaxis.cex = 0.75,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        colourkey.cex = 1,
        colourkey.labels.at = seq(2,12,1),
        colour.alpha = 'automatic',
        grid.row = TRUE,
        grid.col = TRUE,
        row.colour = 'white',
        col.colour = 'white',
        row.lwd = 1.5,
        col.lwd = 1.5,
        # stop heatmap function from rotating matrix
        same.as.matrix = TRUE,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
        );

    # Example of using discrete data
    discrete.data <- microarray[1:10,1:40];
    # Looking for values greater than 10
    discrete.data[which(discrete.data < 10, arr.ind = TRUE)] <- 0;
    discrete.data[which(discrete.data > 0, arr.ind = TRUE)] <- 1;

    sex.colour <- as.character(patient$sex);
    sex.colour[sex.colour == 'male'] <- 'dodgerblue';
    sex.colour[sex.colour == 'female'] <- 'pink';

    stage.colour <- as.character(patient$stage)
    stage.colour[stage.colour == 'I'] <- 'plum1'
    stage.colour[stage.colour == 'II'] <- 'orchid1'
    stage.colour[stage.colour == 'III'] <- 'orchid3'
    stage.colour[stage.colour == 'IV'] <- 'orchid4'

    msi.colour <- as.character(patient$msi)
    msi.colour[msi.colour == 'MSS'] <- 'chartreuse4'
    msi.colour[msi.colour == 'MSI-High'] <- 'chartreuse2'

    discrete.covariate <- list(
        rect = list(
            col = 'transparent',
            fill = sex.colour,
            lwd = 1.5
            ),
        rect = list(
            col = 'transparent',
            fill = stage.colour,
            lwd = 1.5
            ),
        rect = list(
```

```
        col = 'transparent',
        fill = msi.colour,
        lwd = 1.5
        )
    );

discrete.covariate.legend <- list(
    legend = list(
        colours = c('dodgerblue', 'pink'),
        labels = c('male','female'),
        title = expression(underline('Sex'))
        ),
    legend = list(
        colours = c('plum1', 'orchid1', 'orchid3', 'orchid4'),
        labels = c('I','II', 'III', 'IV'),
        title = expression(underline('Stage'))
        ),
    legend = list(
        colours = c('chartreuse4', 'chartreuse2'),
        labels = c('MSS','MSI-High'),
        title = expression(underline('MSI'))
        )
    );

create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Discrete_Data', fileext = '.tiff'),
    x = discrete.data,
    main = 'Discrete data',
    xlab.label = 'Samples',
    same.as.matrix = TRUE,
    # Customize plot
    clustering.method = 'none',
    total.colours = 3,
    colour.scheme = c('white','black'),
    fill.colour = 'grey95',
    # Changing axes
    xat = seq(0,40,5),
    xaxis.lab = seq(0,40,5),
    yaxis.lab = rownames(microarray)[1:10],
    yaxis.cex = 0.75,
    xaxis.cex = 0.75,
    xaxis.rot = 0,
    xlab.cex = 1,
    # Covariates
    covariates.top = discrete.covariate,
    covariate.legend = discrete.covariate.legend,
    legend.side = 'right',
    legend.title.cex = 0.75,
    legend.cex = 0.75,
    legend.title.just = 'left',
    legend.between.row = 0.2,
    legend.border = list(col = 'transparent'),
    legend.border.padding = 2,
```

```
        shrink = 0.7,
        covariates.top.grid.border = list(col = 'black', lwd = 2),
        scale.data = FALSE,
        print.colour.key = FALSE,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
        );

    # Correlation matrix
    # Example of how to visualize the relationship between (e.x.) different cellularity estimates
    # Generate a correlation matrix
    cor.data <- cor(t(microarray[1:10,1:10]), method = 'spearman');
    colnames(cor.data) <- colnames(microarray)[1:10];

    # ensure that input data matrix is equal to what the heatmap clustering produces
    distance.matrix <- as.dist(1 - cor(t(cor.data), use = "pairwise", method = "pearson"));
    clustered.order <- hclust(d = distance.matrix, method = "ward")$order;
    cor.data <- cor.data[clustered.order, clustered.order];

    # prepare labels
    x <- round(cor.data, 2);
    x[x == 1] <- colnames(x);
    y <- x;
    for (i in 1:(ncol(y)-1)) {
        y[i, (i+1):nrow(y)] <- "";
        };

    create.heatmap(
        # filename = tempfile(pattern = 'Heatmap_Cellularity_Estimates', fileext = '.tiff'),
        x = cor.data,
        main = 'Correlation matrix',
        xaxis.lab = NULL,
        yaxis.lab = NULL,
        cell.text = y,
        clustering.method = 'ward',
        plot.dendrograms = 'none',
        rows.distance.method = 'correlation',
        cols.distance.method = 'correlation',
        cor.method = 'pearson',
        col.pos = which(y != '1', arr.ind = TRUE)[,1],
        row.pos = which(y != '1', arr.ind = TRUE)[,2],
        text.fontface = 2,
        text.col = 'white',
        text.cex = 0.70,
        colourkey.cex = 1,
        colour.scheme = c('blue', 'darkgrey', 'brown'),
        colour.centering.value = 0,
        at = seq(-1, 1, 0.001),
        colour.alpha = 1.5,
        grid.row = TRUE,
        grid.col = TRUE,
        description = 'Heatmap created using BoutrosLab.plotting.general',
        resolution = 200
```

```
    );

# Discrete sequential colours
create.heatmap(
   # filename = tempfile(pattern = 'Heatmap_Discrete_Colours_Sequential', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Discrete colours',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    # Adjusting total colours plotted
    colourkey.labels.at = seq(2,12,1),
    at = seq(2,12,1),
    # Add one to account for a 'null' colour
    total.colours = 11,
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Discrete qualitative colours
create.heatmap(
   # filename = tempfile(pattern = 'Heatmap_Discrete_Colours_Qualitative', fileext = '.tiff'),
    x = microarray[1:20, 1:20],
    main = 'Discrete colours',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    # Adjusting total colours plotted
    colourkey.labels.at = seq(2,12,1),
    colourkey.labels = seq(2,12,1),
    at = seq(2,12,1),
    # Add one to account for a 'null' colour
    total.colours = 11,
    colour.scheme = default.colours(10),
    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_Nature_style', fileext = '.tiff'),
```

```
    x = microarray[1:20, 1:20],
    main = 'Nature style',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
    # Adjusting total colours plotted
    colourkey.labels.at = seq(2,12,1),
    colourkey.labels = seq(2,12,1),
    at = seq(2,12,1),
    # Add one to account for a 'null' colour
    total.colours = 11,
    colour.scheme = default.colours(10),

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Heatmap created using BoutrosLab.plotting.general',
    resolution = 200
    );

# create heatmap with key like legend - used to show range of continuous variables

# First create legend with discrete colours
sex.colour <- as.character(patient$sex);
sex.colour[sex.colour == 'male'] <- 'dodgerblue';
sex.colour[sex.colour == 'female'] <- 'pink';

stage.colour <- as.character(patient$stage)
stage.colour[stage.colour == 'I'] <- 'plum1'
stage.colour[stage.colour == 'II'] <- 'orchid1'
stage.colour[stage.colour == 'III'] <- 'orchid3'
stage.colour[stage.colour == 'IV'] <- 'orchid4'

msi.colour <- as.character(patient$msi)
msi.colour[msi.colour == 'MSS'] <- 'chartreuse4'
msi.colour[msi.colour == 'MSI-High'] <- 'chartreuse2'

discrete.covariate <- list(
    rect = list(
        col = 'transparent',
        fill = sex.colour,
        lwd = 1.5
        ),
```

```
    rect = list(
        col = 'transparent',
        fill = stage.colour,
        lwd = 1.5
        ),
    rect = list(
        col = 'transparent',
        fill = msi.colour,
        lwd = 1.5
        )
    );

discrete.covariate.legend <- list(
    legend = list(
        colours = c('dodgerblue', 'pink'),
        labels = c('male','female'),
        title = expression(underline('Sex'))
        ),
    legend = list(
        colours = c('plum1', 'orchid1', 'orchid3', 'orchid4'),
        labels = c('I','II', 'III', 'IV'),
        title = expression(underline('Stage'))
        ),
    legend = list(
        colours = c('chartreuse4', 'chartreuse2'),
        labels = c('MSS','MSI-High'),
        title = expression(underline('MSI'))
        ),
    legend = list(
        colours = c('grey0', 'grey100'),
        labels = c('want key like','legend here'),
        title = expression(underline('one')),
continuous = TRUE,
height=3
        ),
    legend = list(
        colours = c('grey0', 'grey100'),
        labels = c('want key like','legend here'),
        title = expression(underline('two'))
        ),
    legend = list(
        colours = c('grey0', 'grey100'),
        labels = c(0,10),
        title = expression(underline('three')),
continuous = TRUE,
        width = 3,
        tck = 1,
        tck.number = 3,
        at = c(0,100),
angle = -90,
just = c("center","bottom")
        )
   );
```

```
create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_ContinuousVariablesKey', fileext = '.tiff'),
    x = patient[1:20, 4:6],
    xlab.label = 'Samples',
    ylab.label = 'Scaled Data',
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    clustering.method = 'none',
    print.colour.key = FALSE,
    scale=TRUE,
    same.as.matrix = FALSE,
    covariates.top = discrete.covariate,
    covariates.top.grid.row = list(lwd = 1),
    covariate.legends = discrete.covariate.legend,
    legend.title.just = 'left',
    colour.scheme = c('gray0','grey100'),
    fill.colour = 'grey95',
    axis.xlab.padding = 1.5,
    resolution = 200
    );



create.heatmap(
    # filename = tempfile(pattern = 'Heatmap_borderRemoved', fileext = '.tiff'),
    x = simple.data,
    main = 'Simple',
    description = 'Heatmap created using BoutrosLab.plotting.general',
    axes.lwd = 0,
    resolution = 200
    );
```

`create.hexbinplot` *Make a hexagonally binned plot*

Description

Takes a data.frame and writes a hexagonally binned plot

Usage

```
create.hexbinplot(
formula,
data,
filename = NULL,
main = NULL,
main.just = 'center',
```

```
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
        aspect = 'xy',
trans = NULL,
inv = NULL,
colour.scheme = NULL,
colourkey = TRUE,
        colourcut = seq(0, 1, length = 11),
mincnt = 1,
maxcnt = NULL,
xbins = 30,
legend.title = NULL,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
        ylab.cex = 2,
        xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
        xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
        xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.lab = NA,
yaxis.lab = NA,
        xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.col = 'black',
        yaxis.col = 'black',
xaxis.tck = 1,
yaxis.tck = 1,
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
        layout = NULL,
as.table = FALSE,
x.relation = 'same',
y.relation = 'same',
x.spacing = 0,
y.spacing = 0,
strip.col = 'white',
```

```
        strip.cex = 1,
strip.fontface = 'bold',
add.grid = FALSE,
abline.h = NULL,
abline.v = NULL,
abline.lty = NULL,
        abline.lwd = NULL,
abline.col = 'black',
abline.front = FALSE,
add.xyline = FALSE,
xyline.col = 'black',
        xyline.lwd = 1,
xyline.lty = 1,
add.curves = FALSE,
curves.exprs = NULL,
curves.from = min(data, na.rm = TRUE),
        curves.to = max(data, na.rm = TRUE),
curves.col = 'black',
curves.lwd = 2,
curves.lty = 1,
        add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.col = 'black',
text.cex = 1,
        text.fontface = 'bold',
add.axes = FALSE,
top.padding = 0.1,
bottom.padding = 0.7,
        left.padding = 0.5,
right.padding = 0.1,
        add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
        ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
        background.col = 'transparent',
key = NULL,
legend = NULL,
        height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
        description = 'Created with BoutrosLab.plotting.general',
```

```
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE,
        inside.legend.auto = FALSE
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame. Transforming data within formula is not compatible with automatic scaling with ‘xat‘ or ‘yat‘.
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL (default value) returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of the main plot title
`aspect`	This argument controls the physical aspect ratio of the panels, defaults to “xy”
`trans`	function specifying a transformation for the counts such as log, defaults to NULL
`inv`	the inverse transformation of trans, defaults to NULL
`colour.scheme`	colour scheme to be used, default NULL gives LinGray colour scale
`colourkey`	logical whether a legend should be drawn, defaults to TRUE
`colourcut`	Vector of values covering [0, 1] that determine hexagon colour class boundaries and hexagon legend size boundaries. Alternatively, an integer (<= maxcnt) specifying the number of equispaced colourcut values in [0,1].
`mincnt`	Cells with fewer counts are ignored
`maxcnt`	Cells with more counts are ignored, defaults to auto-generation
`xbins`	Number of bins to use in x, defaults to 30
`legend.title`	character/expression to use in place of default legend title or a named list with elements: lab, x, y; defaults to NULL
`xlab.label`	X-axis label
`ylab.label`	Y-axis label
`xlab.cex`	Size of x-axis label
`ylab.cex`	Size of y-axis label
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label

`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits
`ylimits`	Two-element vector giving the y-axis limits
`xat`	Accepts a vector listing where x-axis ticks should be drawn or if automatic scaling is desired, one of three strings: "auto", "auto.linear" or "auto.log". Automatic scaling fixes x-axis tick locations, labels, and data values dependent given data. "auto" will determine whether linear or logarithmic scaling fits the given data best, "auto.linear" or "auto.log" will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see 'auto.axis()'.
`yat`	Accepts a vector listing where y-axis ticks should be drawn or if automatic scaling is desired, one of three strings: "auto", "auto.linear" or "auto.log". Automatic scaling fixes y-axis tick locations, labels, and data values dependent given data. "auto" will determine whether linear or logarithmic scaling fits the given data best, "auto.linear" or "auto.log" will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see 'auto.axis()'.
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with xat will overwrite user input. Set to NULL to remove x-axis labels.
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with yat will overwrite user input. Set to NULL to remove y-axis labels.
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.col`	Colour of the x-axis tick labels, defaults to "black"
`yaxis.col`	Colour of the y-axis tick labels, defaults to "black"
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL.
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.relation`	Allows x-axis scales to vary if set to "free", defaults to "same"
`y.relation`	Allows y-axis scales to vary if set to "free", defaults to "same"
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0

`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`strip.fontface`	Strip title fontface, defaults to bold
`add.grid`	Allows grid lines to be turned on or off
`abline.h`	Specify the superimposed horizontal line(s)
`abline.v`	Specify the superimposed vertical line(s)
`abline.lty`	Specify the superimposed line type
`abline.lwd`	Specify the superimposed line width
`abline.col`	Specify the superimposed line colour (defaults to black)
`abline.front`	If an abline and/or a grid has been added, this controls whether they are drawn in front of the hexbins
`add.xyline`	Allow y=x line to be drawn, default is FALSE
`xyline.col`	y=x line colour, defaults to black
`xyline.lwd`	Specifies y=x line width, defaults to 1
`xyline.lty`	Specifies y=x line style, defaults to 1 (solid)
`add.curves`	Allow curves to drawn, default is FALSE
`curves.exprs`	A list of functions, expressions, or calls using “x” as a variable that specify the curves to be drawn
`curves.from`	Specifies the x co-ordinates at which the start of each curve should be drawn, defaults to drawing the curves to the left edge of the plotting region
`curves.to`	Specifies the x co-ordinates at which the end of each curve should be drawn, defaults to drawing the curves to the right edge of the plotting region
`curves.col`	Specifies colours of curves, default is black for each curve
`curves.lwd`	Specifies width of curves, default is 1 for each curve
`curves.lty`	Specifies type of curves, default is 1 (solid) for each curve
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`add.axes`	Allow axis lines to be turned on or off
`top.padding`	A number giving the top padding in multiples of the lattice default
`bottom.padding`	A number giving the bottom padding in multiples of the lattice default
`left.padding`	A number giving the left padding in multiples of the lattice default
`right.padding`	A number giving the right padding in multiples of the lattice default

`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`background.col`	Specifies the colour for the background of the plot
`key`	Add a key to the plot. See xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Details

WARNING: this function uses highly unusual semantics, different from the rest of the `BoutrosLab.plotting.general` library. The underlying hexbinplot function uses an argument called maxcnt to specify the maximum number of counts per cell. The default behaviour is not sensibly encoded via a `NULL` or an `NA`, but instead by using the `missing` function. As a result, we need to use `do.call` semantics to handle this function. This **can mess up anything using** `substitute` including things that generate p-values!

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
    Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
        Invalid font type
    Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
If 'maxcnt' is passed, make sure it is not smaller than the actual maximum count (value depends on nbins)
```

Author(s)

Maud HW Starmans

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = rnorm(10000),
    y = rnorm(10000)
    );

create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Simple', fileext = '.tiff'),
    formula = y ~ x,
    data = simple.data,
    main = 'Simple',
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

create.hexbinplot(
   # filename = tempfile(pattern = 'Hexbinplot_Simple_underlined_legend_title', fileext = '.tiff'),
    formula = y ~ x,
    data = simple.data,
    legend.title = list(lab = expression(bold(underline('Counts'))), x = 1, y = 1.1),
    right.padding = 4,
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Set up data
hexbin.data <- data.frame(
```

```
    x = microarray[,1],
    y = microarray[,2]
    );

# Minimal Input
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Minimal_Input', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Minimal input',
    # formatting bins
    colourcut = seq(0, 1, length = 11),
    # this sets the maximum value plotted -- values greater than this will not appear
    maxcnt = 50,
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes & Labels
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Axes_Labels', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Axes & labels',
    colourcut = seq(0, 1, length = 11),
    maxcnt = 50,
    # Customize Axes and labels
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlimits = c(0,16),
    ylimits = c(0,16),
    xat = seq(0,16,2),
    yat = seq(0,16,2),
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Log Scaled Axis
log.data <- data.frame(
    x = microarray[,1],
    y = 10 ** microarray[,2]
    );

create.hexbinplot(
    formula = y ~ x,
    data = log.data,
    main = 'Log Scaled',
```

```
    # Log base 10 scale y-axis
    yat = 'auto.log',
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Aspect Ratio
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Aspect_Ratio', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Aspect ratio',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlimits = c(0,16),
    ylimits = c(0,16),
    xat = seq(0,16,2),
    yat = seq(0,16,2),
    colourcut = seq(0, 1, length = 11),
    maxcnt = 50,
    # Set the aspect ratio to control plot dimensions
    aspect = 2,
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Colour scheme
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Colour_Change', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Colour change',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlimits = c(0,16),
    ylimits = c(0,16),
    xat = seq(0,16,2),
    yat = seq(0,16,2),
    aspect = 1,
    colourcut = seq(0, 1, length = 11),
```

```
        maxcnt = 50,
        # Specify colour scheme
       colour.scheme = colorRampPalette(c('dodgerblue','paleturquoise','chartreuse','yellow',
            'orange','red')),
        description = 'Hexbinplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Bin sizes
    create.hexbinplot(
        # filename = tempfile(pattern = 'Hexbinplot_Bin_Sizes', fileext = '.tiff'),
        formula = y ~ x,
        data = hexbin.data,
        main = 'Bin sizes',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xlab.label = 'Sample 1',
        ylab.label = 'Sample 2',
        xlimits = c(0,16),
        ylimits = c(0,16),
        xat = seq(0,16,2),
        yat = seq(0,16,2),
        aspect = 1,
       colour.scheme = colorRampPalette(c('dodgerblue','paleturquoise','chartreuse', 'yellow',
            'orange','red')),
        # Specify bin sizes
        colourcut = seq(0,1,length = 6),
        description = 'Hexbinplot created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Correlation Key
    create.hexbinplot(
        # filename = tempfile(pattern = 'Hexbinplot_Correlation', fileext = '.tiff'),
        formula = y ~ x,
        data = hexbin.data,
        main = 'Correlation',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xlab.label = 'Sample 1',
        ylab.label = 'Sample 2',
        xlimits = c(0,16),
        ylimits = c(0,16),
        xat = seq(0,16,2),
        yat = seq(0,16,2),
```

```
    aspect = 1,
    colourcut = seq(0, 1, length = 11),
    maxcnt = 50,
    # Correlation Key
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = get.corr.key(
                    x = hexbin.data$x,
                    y = hexbin.data$y,
                    label.items = c('beta1', 'spearman'),
                    alpha.background = 0
                    )
                ),
            x = 0.05,
            y = 0.95,
            corner = c(0,1),
            draw = FALSE
            )
        ),
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Grid lines and diagonal
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Gridlines', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Gridlines',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlimits = c(0,16),
    ylimits = c(0,16),
    xat = seq(0,16,2),
    yat = seq(0,16,2),
    aspect = 1,
    colourcut = seq(0, 1, length = 11),
    maxcnt = 50,
    # Grid & diagonal
    add.grid = TRUE,
    add.xyline = TRUE,
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# Large range
# Generate some fake data with both very low and very high values
set.seed(12345);

x <- c(rnorm(100000,0,0.1),rnorm(1000,0,0.5),rnorm(1000,0,sd=0.75));
y <- c(rnorm(100000,0,0.1),rnorm(1000,0,0.5),rnorm(1000,0,sd=0.75));

fake.data <- data.frame(
    x = x,
    y = y,
    z = y + x*(x+1)/4
    );

create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Range', fileext = '.tiff'),
    formula = z ~ x,
    data = fake.data,
    main = 'Range',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    aspect = 1,
    # Use colourcut to divide the bins appropriately
   colourcut = c(0,0.0002,0.0004,0.0008,0.0016,0.0032,0.0064,0.0128,0.0256,0.0512,0.1024,0.2048,
        0.4096,0.8192,1),
    # Change the colour scheme
    colour.scheme = function(n){BTC(n, beg=1, end=256)},
    background.col = 'grey',
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Outliers
# Generate data with upper bound outlier
set.seed(12345);

x <- c(rnorm(1000,0,0),rnorm(4000,0,0.5));
y <- c(rnorm(1000,0,0),rnorm(4000,0,0.5));

fake.data.outlier <- data.frame(
    x = x,
    y = y,
    z = y + x*(x+1)/4
    );

create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Outlier', fileext = '.tiff'),
    formula = z ~ x,
```

```
    data = fake.data.outlier,
    main = 'Outlier',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    aspect = 1,
    # Use colourcut to divide the bins appropriately
   colourcut = c(seq(0,0.01, length = 4),seq(0.0125,0.1,length=4), seq(0.125,1,length=4)),
    xbins = 15,
    mincnt = 0,
    # Change the colour scheme
    colour.scheme = function(n){BTC(n, beg=1, end=256)},
    background.col = 'grey',
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.hexbinplot(
    # filename = tempfile(pattern = 'Hexbinplot_Nature_style', fileext = '.tiff'),
    formula = y ~ x,
    data = hexbin.data,
    main = 'Nature style',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,16),
    ylimits = c(0,16),
    xat = seq(0,16,2),
    yat = seq(0,16,2),
    aspect = 1,
    colourcut = seq(0, 1, length = 11),
    maxcnt = 50,
    # Grid & diagonal
    add.grid = TRUE,
    add.xyline = TRUE,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),
```

```
    description = 'Hexbinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Multiplot different groups
set.seed(73);

# Randomly generate groups
simple.data$groups <- sample(1:2, 10000, replace = TRUE);
simple.data$group.labels <- as.factor(simple.data$groups);

create.hexbinplot(
formula = y ~ x | groups,
# filename = tempfile(
  #      pattern = 'stratified_hexbinplot_numeric_conditioning',
  #      fileext = '.tiff'
  #      ),
data = simple.data,
description = 'Hexbinplot created by BoutrosLab.plotting.general',
strip.col = 'white',
strip.cex = 0.8,
strip.fontface = 'bold',
resolution = 200
);

create.hexbinplot(
formula = y ~ x | group.labels,
# filename = tempfile(
  #      pattern = 'stratified_hexbinplot_factor_conditioning',
  #      fileext = '.tiff'
  #      ),
data = simple.data,
description = 'Hexbinplot created by BoutrosLab.plotting.general',
strip.col = 'white',
strip.cex = 0.8,
strip.fontface = 'bold',
resolution = 200
);
```

`create.histogram` *Make a histogram*

Description

Takes a vector and creates a histogram

Usage

```
create.histogram(
x,
data,
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = NULL,
ylab.label = NULL,
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xaxis.lab = TRUE,
yaxis.lab = TRUE,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.tck = 1,
yaxis.tck = 1,
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
type = 'percent',
breaks = NULL,
col = 'white',
border.col = 'black',
lwd = 2,
lty = 1,
layout = NULL,
x.spacing = 0,
y.spacing = 0,
x.relation = 'same',
```

```
y.relation = 'same',
strip.col = 'white',
strip.cex = 1,
top.padding = 0.1,
bottom.padding = 0.7,
right.padding = 0.1,
left.padding = 0.5,
ylab.axis.padding = 0,
abline.h = NULL,
abline.v = NULL,
abline.col = 'black',
abline.lwd = 1,
abline.lty = 1,
key = NULL,
legend = NULL,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
        use.legacy.settings = FALSE,
inside.legend.auto = FALSE
);
```

Arguments

`x`	A formula or a numeric vector (not frequencies!)
`data`	An optional data source if x is a formula
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered

`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 2
`xlab.label`	x-axis title
`ylab.label`	y-axis title
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis tick labels, defaults to 1
`yaxis.cex`	Size of y-axis tick labels, defaults to 1
`xlimits`	Two-element vector giving the x-axis limits
`ylimits`	Two-element vector giving the y-axis limits
`xat`	Vector listing where the x-axis ticks should be drawn
`yat`	Vector listing where the y-axis ticks should be drawn
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`type`	Should the plot be of the “percent” (default), “density” or “count”
`breaks`	A vector listing the break-points of the histogram, or an integer specifying the desired number of breaks.
`col`	Fill colour for the histograms
`border.col`	Specify border colour (defaults to black)
`lwd`	Specifies line width

`lty`	Specifies line style
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details

.

`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.5
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 0

,

`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.v`	Allow vertical line to be drawn, default to NULL
`abline.col`	Horizontal and vertical line colour, defaults to black
`abline.lwd`	Specifies horizontal/vertical line width, defaults to 1
`abline.lty`	Specifies horizontal/vertical line style, defaults to 1 (solid)
`key`	Add a key to the plot. See xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn

`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`histogram`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

create.histogram(
    # filename = tempfile(pattern = 'Histogram_Simple', fileext = '.tiff'),
    x = rnorm(5000),
    main = 'Simple',
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 50
    );

create.histogram(
    # filename = tempfile(pattern = 'Histogram_Simple_Count', fileext = '.tiff'),
    x = rnorm(5000),
    main = 'Simple Count',
    description = 'Histogram created by BoutrosLab.plotting.general',
    type = 'count',
    resolution = 50
    );

# Minimal Input
create.histogram(
    # filename = tempfile(pattern = 'Histogram_Minimal_Input', fileext = '.tiff'),
    x = microarray[,1],
    main = 'Minimal input',
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Formula Input - dividing by chromosome
chr.data <- data.frame(
    x = microarray$Chr,
    y = microarray[,1]
    );

create.histogram(
    # filename = tempfile(pattern = 'Histogram_Formula_Input', fileext = '.tiff'),
    x = y ~ x,
    data = chr.data,
    main = 'Formula input',
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes and Labels
create.histogram(
    # filename = tempfile(pattern = 'Histogram_Axes_Labels', fileext = '.tiff'),
    x = microarray[,1],
    main = 'Axes & labels',
    # Customizing the axes and labels
    xlab.label = 'Bins',
    ylab.label = 'Counts',
```

```
    xlimits = c(0, 16),
    xat = seq(0,15,5),
    # set break points for bins
    breaks = seq(floor(min(microarray[,1])), ceiling(max(microarray[,1])), 1),
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Colour change
create.histogram(
    # filename = tempfile(pattern = 'Histogram_Colours', fileext = '.tiff'),
    x = microarray[,1],
    main = 'Colours',
    xlab.label = 'Bins',
    ylab.label = 'Counts',
    xlimits = c(0, 16),
    xat = seq(0,15,5),
    breaks = seq(floor(min(microarray[,1])), ceiling(max(microarray[,1])), 1),
    # Colours
    col = 'lightgrey',
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Line type
create.histogram(
    # filename = tempfile(pattern = 'Histogram_Line_Type', fileext = '.tiff'),
    x = microarray[,1],
    main = 'Line type',
    xlab.label = 'Bins',
    ylab.label = 'Counts',
    xlimits = c(0, 16),
    xat = seq(0,15,5),
    breaks = seq(floor(min(microarray[,1])), ceiling(max(microarray[,1])), 1),
    col = 'lightgrey',
    # Changing the line type
    lty = 2,
    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.histogram(
    # filename = tempfile(pattern = 'Histogram_Nature_style', fileext = '.tiff'),
    x = microarray[,1],
    main = 'Nature style',
    xlimits = c(0, 16),
    xat = seq(0,15,5),
    breaks = seq(floor(min(microarray[,1])), ceiling(max(microarray[,1])), 1),
    col = 'lightgrey',

    # set style to Nature
```

```
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Histogram created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

`create.lollipopplot` *Make a lollipopplot*

Description

Takes a data.frame and creates a lollipopplot

Usage

```
create.lollipopplot(
formula,
data,
filename = NULL,
groups = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
        main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
        xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
        xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
        xlab.top.y = 0,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
```

```
xaxis.lab = NA,
yaxis.lab = NA,
xaxis.log = FALSE,
yaxis.log = FALSE,
        xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.tck = c(1,1),
yaxis.tck = c(1,1),
add.grid = FALSE,
xgrid.at = xat,
        ygrid.at = yat,
grid.colour = NULL,
horizontal = FALSE,
type = 'p',
cex = 0.75,
pch = 19,
col = 'black',
col.border = 'black',
lwd = 1,
lty = 1,
alpha = 1,
axes.lwd = 1,
strip.col = 'white',
strip.cex = 1,
strip.fontface = 'bold',
y.error.up = NULL,
y.error.down = y.error.up,
x.error.right = NULL,
x.error.left = x.error.right,
y.error.bar.col = 'black',
x.error.bar.col = y.error.bar.col,
error.whisker.angle = 90,
error.bar.lwd = 1,
error.bar.length = 0.1,
key = list(text = list(lab = c(''))),
legend = NULL,
top.padding = 0.1,
bottom.padding = 0.7,
right.padding = 0.1,
left.padding = 0.5,
key.top = 0.1,
key.left.padding = 0,
```

```
  ylab.axis.padding = 1,
  axis.key.padding = 1,
  layout = NULL,
  as.table = FALSE,
  x.spacing = 0,
  y.spacing = 0,
  x.relation = 'same',
  y.relation = 'same',
  add.axes = FALSE,
  axes.lty = 'dashed',
  add.xyline = FALSE,
  xyline.col = 'black',
  xyline.lwd = 1,
  xyline.lty = 1,
  abline.h = NULL,
  abline.v = NULL,
  abline.col = 'black',
  abline.lwd = 1,
  abline.lty = 1,
  add.curves = FALSE,
  curves.exprs = NULL,
  curves.from = min(data, na.rm = TRUE),
  curves.to = max(data, na.rm = TRUE),
  curves.col = 'black',
  curves.lwd = 2,
  curves.lty = 1,
  add.rectangle = FALSE,
  xleft.rectangle = NULL,
  ybottom.rectangle = NULL,
  xright.rectangle = NULL,
  ytop.rectangle = NULL,
  col.rectangle = 'transparent',
  alpha.rectangle = 1,
  add.points = FALSE,
  points.x = NULL,
  points.y = NULL,
  points.pch = 19,
  points.col = 'black',
  points.col.border = 'black',
  points.cex = 1,
  add.line.segments = FALSE,
  line.start = NULL,
  line.end = NULL,
  line.col = 'black',
  line.lwd = 1,
  add.text = FALSE,
  text.labels = NULL,
  text.x = NULL,
```

```
text.y = NULL,
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
text.guess.labels = FALSE,
text.guess.skip.labels = TRUE,
text.guess.ignore.radius = FALSE,
text.guess.ignore.rectangle = FALSE,
text.guess.radius.factor = 1,
text.guess.buffer.factor = 1,
text.guess.label.position = NULL,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
group.specific.colouring = TRUE,
use.legacy.settings = FALSE,
inside.legend.auto = FALSE,
regions.labels = c(),
        regions.start = c(),
regions.stop = c(),
regions.color = c("red"),
regions.cex = 1,
regions.alpha = 1,
        lollipop.bar.y = NULL,
lollipop.bar.color = "gray",
...
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`groups`	The grouping variable in the data-frame
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title
`xlab.label`	x-axis label
`ylab.label`	y-axis label

`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2
`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`yaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`add.grid`	Logical stating wheter or not the grid should be drawn on the plot
`xgrid.at`	Vector listing where the x-axis grid lines should be drawn, defaults to xat
`ygrid.at`	Vector listing where the y-axis grid lines should be drawn, defaults to yat
`grid.colour`	ability to set individual grid line colours
`horizontal`	xyplot-specific function that allows you to change if type=’h’ draws lines to the vertical or horizontal axis
`type`	Plot type
`cex`	Character expansion for plotting symbol

`pch`	Plotting character
`col`	Point/line colour
`col.border`	Colour of border when points pch >= 21. Defaults to “black”
`lwd`	Specifies line width, defaults to 1
`lty`	Specifies line style, defaults to 1 (solid)
`alpha`	Specifies line transparency, defaults to 1 (opaque)
`axes.lwd`	Thickness of width of axes lines
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`strip.fontface`	Strip title fontface, defaults to bold
`y.error.up`	upward error vector. Defaults to NULL. When y.error.up is NULL, vertical error bar is not drawn
`y.error.down`	Downward error vector. Defaults to y.error.down to show symmetric error bars
`x.error.right`	Rightward error vector. Defaults to NULL. When x.error.right is NULL, horizontal error bar is not drawn
`x.error.left`	Leftward error vector. Defaults to x.error.right to show symmetric error bars
`y.error.bar.col`	Colour of vertical error bar. Defaults to “black”
`x.error.bar.col`	Colour of horizontal error bar. Defaults to “black”
`error.whisker.angle`	Angle of the whisker drawn on error bar. Defaults to 90 degree
`error.bar.lwd`	Error bar line width. Defaults to 1
`error.bar.length`	Length of the error bar whiskers. Defaults to 0.1
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.1
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`key.top`	A number specifying the distance at top of key, defaults to 0.1
`key.left.padding`	Amount of padding to go onto any legend on the left
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 1
`axis.key.padding`	A number specifying the distance from the y-axis to the key, defaults to 1
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details

`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`add.axes`	Allow axis lines to be turned on or off, default is FALSE
`axes.lty`	Specifies axis line style, defaults to “dashed”
`add.xyline`	Allow y=x line to be drawn, default is FALSE
`xyline.col`	y=x line colour, defaults to black
`xyline.lwd`	Specifies y=x line width, defaults to 1
`xyline.lty`	Specifies y=x line style, defaults to 1 (solid)
`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.v`	Allow vertical line to be drawn, default to NULL
`abline.col`	Horizontal line colour, defaults to black
`abline.lwd`	Specifies horizontal line width, defaults to 1
`abline.lty`	Specifies horizontal line style, defaults to 1 (solid)
`add.curves`	Allow curves to drawn, default is FALSE
`curves.exprs`	A list of functions, expressions, or calls using “x” as a variable that specify the curves to be drawn
`curves.from`	Specifies the x co-ordinates at which the start of each curve should be drawn, defaults to drawing the curves to the left edge of the plotting region
`curves.to`	Specifies the x co-ordinates at which the end of each curve should be drawn, defaults to drawing the curves to the right edge of the plotting region
`curves.col`	Specifies colours of curves, default is black for each curve
`curves.lwd`	Specifies width of curves, default is 1 for each curve
`curves.lty`	Specifies type of curves, default is 1 (solid) for each curve
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`add.points`	Allow additional points to be drawn, default is FALSE

`points.x`	The x co-ordinates where additional points should be drawn
`points.y`	The y co-ordinates where additional points should be drawn
`points.pch`	The plotting character for additional points
`points.col`	The colour of additional points
`points.col.border`	Colour of the border of additional points if points.pch >= 21. Defaults to black
`points.cex`	The size of additional points
`add.line.segments`	Allow additional line segments to be drawn, default is FALSE
`line.start`	The y co-ordinates where additional line segments should start
`line.end`	The y co-ordinates where additional line segments should end
`line.col`	The colour of additional line segments, default is black
`line.lwd`	The line width of additional line segments, default is 1
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`text.guess.labels`	Allows automatic labeling by considering values in text.x and text.y as a data point to be labelled, default is FALSE
`text.guess.skip.labels`	Provides an option to disregard automatic labelling algorithm if no space is available around a data point, thus forcing labelling if a collision is likely, default is TRUE
`text.guess.ignore.radius`	Allows the automatic labeling algorithm to ignore the radius space of a data point, useful to label a cluster of data points with a single text box, default is FALSE
`text.guess.ignore.rectangle`	Allows the atuomatic labeling algorithm to ignore the rectangle space of multiple potential label positions, default is FALSE
`text.guess.radius.factor`	A numeric value to factor the radius value to alter distance from the label and the data point
`text.guess.buffer.factor`	A numeric value to factor the buffer value to alter the space which is used to consider if data.points are potentially going to collide
`text.guess.label.position`	A numeric value between 0 and 360 to specify the percise angle of a text box center and the positive x-axis. Angles move counter-clockwise beginning at the positive x axis

`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`group.specific.colouring`	Variable to specify if group specific multi colouring for error bars is enforced
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function
`regions.labels`	Labels for each of the regions on the lollipop plots bars
`regions.start`	start x value of each of the regions
`regions.stop`	stop value for each of the regions
`regions.color`	color of each of the regions
`regions.cex`	size of the text of each of the regions
`regions.alpha`	alpha of each of the regions
`lollipop.bar.y`	y location of top of the lollipop plot bar – defaults to right above the bottom y axis
`lollipop.bar.color`	color of the lollipop plot bar
`...`	Additional arguments to be passed to xyplot

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);
lollipop.data <- data.frame(
    y = seq(1,100,1),
    x = rnorm(100)
    );

create.lollipopplot(
    # filename = tempfile(pattern = 'Lollipop_Simple', fileext = '.tiff'),
    formula = x ~ y,
    data = lollipop.data,
    main = 'Lollipop plot',
    xaxis.cex = 1,
    xlimits = c(-1,102),
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    regions.start = c(1,26,48),
    regions.stop = c(15,35,72),
    regions.labels = c("test 1", "test2", "test 3"),
    regions.color = c("#66b3ff", "#5cd65c", "#ff3333")
    );
```

`create.manhattanplot` *Make a Manhattan plot*

Description

Takes a data.frame and creates a Manhattan plot

Usage

```
create.manhattanplot(
formula,
data,
```

```
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.lab = NA,
yaxis.lab = NA,
xaxis.log = FALSE,
yaxis.log = FALSE,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.fontface = 'plain',
yaxis.fontface = 'plain',
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.tck = 0,
yaxis.tck = c(1,1),
horizontal = FALSE,
type = 'p',
cex = 2,
pch = '.',
col = 'black',
lwd = 1,
lty = 1,
alpha = 1,
strip.col = 'white',
strip.cex = 1,
axes.lwd = 1,
axes.lty = 'dashed',
```

```
key = list(text = list(lab = c(''))),
legend = NULL,
layout = NULL,
as.table = FALSE,
x.spacing = 0,
y.spacing = 0,
x.relation = 'same',
y.relation = 'same',
top.padding = 0,
bottom.padding = 0,
right.padding = 0,
left.padding = 0,
key.top = 0,
key.left.padding = 0,
ylab.axis.padding = 1,
axis.key.padding = 1,
abline.h = NULL,
abline.col = 'black',
abline.lwd = 1,
abline.lty = 1,
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
add.points = FALSE,
points.x = NULL,
points.y = NULL,
points.pch = 19,
points.col = 'black',
points.cex = 1,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
height = 6,
width = 10,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
style = 'BoutrosLab',
description = 'Created with BoutrosLab.plotting.general',
preload.default = 'custom',
```

```
        use.legacy.settings = FALSE,
        inside.legend.auto = FALSE,
...
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame. Transforming data within formula is not compatible with automatic scaling with ‘xat‘ or ‘yat‘.
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title
`xlab.label`	x-axis label
`ylab.label`	y-axis label
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Accepts a vector listing where x-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes x-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.

`yat`	Accepts a vector listing where y-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes y-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with xat will overwrite user input. Set to NULL to remove x-axis labels.
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with yat will overwrite user input. Set to NULL to remove y-axis labels.
`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.cex`	Size of x-axis scales, defaults to 1
`yaxis.cex`	Size of y-axis scales, defaults to 1
`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`yaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`horizontal`	xyplot-specific function that allows you to change if type=’h’ draws lines to the vertical or horizontal axis
`type`	Plot type
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`col`	Point/line colour
`lwd`	Specifies line width, defaults to 1
`lty`	Specifies line style, defaults to 1 (solid)
`alpha`	Specifies line transparency, defaults to 1 (opaque)
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`axes.lwd`	Thickness of width of axis lines
`axes.lty`	Specifies axis line style, defaults to “dashed”
`key`	A list giving the key (legend). The default suppresses drawing

`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
F	
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`top.padding`	A number specifying the distance to the top margin, defaults to 0
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0
`right.padding`	A number specifying the distance to the right margin, defaults to 0
`left.padding`	A number specifying the distance to the left margin, defaults to 0
`key.top`	A number specifying the distance at top of key, defaults to 0
`key.left.padding`	Amount of padding to go onto any legend on the left
`ylab.axis.padding`	A number specifying the distance of label to the y-axis, defaults to 1
`axis.key.padding`	A number specifying the distance from the y-axis to the key, defaults to 1
`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.col`	Horizontal line colour, defaults to black
`abline.lwd`	Specifies horizontal line width, defaults to 1
`abline.lty`	Specifies horizontal line style, defaults to 1 (solid)
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`add.points`	Allow additional points to be drawn, default is FALSE
`points.x`	The x co-ordinates where additional points should be drawn

`points.y`	The y co-ordinates where additional points should be drawn
`points.pch`	The plotting character for additional points
`points.col`	The colour of additional points
`points.cex`	The size of additional points
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`description`	Short description of image/plot; default NULL
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function
`...`	Additional arguments to be passed to xyplot

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Christine P'ng, Cindy Q. Yao

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);
simple.data <- data.frame(
    x = runif(20000, 0, 1),
    y = 1:20000
    );

create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Simple', fileext = '.tiff'),
    formula = -log10(x) ~ y,
    data = simple.data,
    main = 'Simple',
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 50
    );

# set up chromosome covariate colours to use for chr covariate, below
chr.colours <- force.colour.scheme(microarray$Chr, scheme = 'chromosome');

# make chr covariate and chr labels
chr.n.genes       <- vector();
chr.tck           <- vector();
chr.pos.genes     <- vector();
chr.break         <- vector();
chr.break[1]      <- 0;
# get a list of chromosomes to loop
chr <- unique(microarray$Chr);

# loop over each chromosome
for ( i in 1:length(chr) ) {

    # get the number of genes that belong to one chromosome
    n <- sum(microarray$Chr == chr[i]);

    # calculate where the labels go
    chr.n.genes[i]   <- n;
    chr.break[i+1] <- n + chr.break[i];
    chr.pos.genes[i] <- floor(chr.n.genes[i]/2);
    chr.tck[i]       <- chr.pos.genes[i] + which(microarray$Chr == chr[i])[1];
    }

# add an indicator function for the data-frame
microarray$ind <- 1:nrow(microarray);
```

```
# Minimal input
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Minimal_Input', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Minimal input',
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 100
    );

# Custom Axes
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Custom_Axes', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Custom axes',
    xlab.label = expression('Chromosomes'),
    ylab.label = expression('P'['adjusted']),
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
        expression(10^-4)
        ),
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 100
    );

# Log-Scaled Axis
log.data <- data.frame(
    x = 10 ** runif(20000, 1, 5),
    y = 1:20000
    );

create.manhattanplot(
    formula = x ~ y,
    data = log.data,
    main = 'Log Scaled',
    # Log base 10 scale x-axis
    xat = 'auto.log',
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 50
    );

# Colour scheme
create.manhattanplot(
```

```
    # filename = tempfile(pattern = 'Manhattan_Colour_Scheme', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Colour scheme',
    xlab.label = expression('Chromosomes'),
    ylab.label = expression('P'['adjusted']),
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
        expression(10^-4)
        ),
    col = chr.colours,
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Plotting Character
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Plotting_Character', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Plotting character',
    xlab.label = expression('Chromosomes'),
    ylab.label = expression('P'['adjusted']),
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
        expression(10^-4)
        ),
    col = chr.colours,
    # Change plotting character and size of plotting character
    pch = 18,
    cex = 0.75,
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# Line
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Added_Line', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Line',
    xlab.label = expression('Chromosomes'),
    ylab.label = expression('P'['adjusted']),
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
        expression(10^-4)
        ),
    col = chr.colours,
    pch = 18,
    cex = 0.75,
    # draw horizontal line
    abline.h = 2,
    abline.lty = 2,
    abline.lwd = 1,
    abline.col = 'black',
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Background shading
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_BG', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Bg rectangles',
    xlab.label = expression('Chromosomes'),
    ylab.label = expression('P'['adjusted']),
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
```

```
        expression(10^-4)
        ),
    col = chr.colours,
    pch = 18,
    cex = 0.75,
    abline.h = 2,
    abline.lty = 2,
    abline.lwd = 1,
    abline.col = 'black',
    # Adding rectangles
    add.rectangle = TRUE,
    xleft.rectangle = chr.break[seq(1, length(chr.break) - 1, 2)],
    ybottom.rectangle = 0,
    xright.rectangle = chr.break[seq(2, length(chr.break) - 1, 2)],
    ytop.rectangle = 4.5,
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,
    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.manhattanplot(
    # filename = tempfile(pattern = 'Manhattan_Nature_style', fileext = '.tiff'),
    formula = -log10(pval) ~ ind,
    data = microarray,
    main = 'Nature style',
    xat = chr.tck,
    xaxis.lab = c(1:22, 'X', 'Y'),
    xaxis.tck = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    yat = seq(0,5,1),
    yaxis.lab = c(
        1,
        expression(10^-1),
        expression(10^-2),
        expression(10^-3),
        expression(10^-4)
        ),
    col = chr.colours,
    pch = 18,
    cex = 0.75,
    abline.h = 2,
    abline.lty = 2,
    abline.lwd = 1,
    abline.col = 'black',
    # Adding rectangles
    add.rectangle = TRUE,
    xleft.rectangle = chr.break[seq(1, length(chr.break) - 1, 2)],
    ybottom.rectangle = 0,
    xright.rectangle = chr.break[seq(2, length(chr.break) - 1, 2)],
    ytop.rectangle = 4.5,
```

```
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Manhattan plot created using BoutrosLab.plotting.general',
    resolution = 1200
    );
```

`create.multipanelplot` *Joins plots together*

Description

Merges together multiple plots in the specified layout

Usage

```
create.multipanelplot(
plot.objects = NULL,
filename = NULL,
height = 10,
width = 10,
resolution = 1000,
plot.objects.heights = c(rep(1,layout.height)),
plot.objects.widths = c(rep(1,layout.width)),
layout.width = 1,
layout.height = length(plot.objects),
main = '',
main.x = 0.5,
main.y = 0.5,
x.spacing = 0,
y.spacing = 0,
xlab.label = '',
xlab.cex = 2,
ylab.label = '',
ylab.label.right = '',
ylab.cex = 2,
main.cex = 3,
legend = NULL,
```

```
left.padding = 0,
ylab.axis.padding = c(rep(0, layout.width)),
xlab.axis.padding = c(rep(0, layout.height)),
bottom.padding = 0,
top.padding = 0,
right.padding = 0,
layout.skip = c(rep(FALSE, layout.width*layout.height)),
left.legend.padding = 2,
right.legend.padding = 2,
bottom.legend.padding = 2,
top.legend.padding = 2,
description = 'Created with BoutrosLab.plotting.general',
size.units = 'in',
enable.warnings = FALSE,
style = "BoutrosLab",
use.legacy.settings = FALSE
);
```

Arguments

`plot.objects`	A list of plot objects. Goes in this order: Top Left, Top Right, Bottom Left, Bottom Right
`filename`	Filename to output to
`height`	Height of resulting file
`width`	Width of resulting file
`resolution`	Resolution of resulting file
`plot.objects.heights`	Heights of each row of the plot. Must be vector of same size as layout.height
`plot.objects.widths`	Widths of each column of the plot. Must be vector of same size as layout.width
`layout.width`	how many plots per row.
`layout.height`	how many plots per column
`main`	main label text
`main.x`	main label x coordinate
`main.y`	main label y coordinate
`x.spacing`	horizontal spacing between each plot. Can be single value or vector of length layout.width - 1
`y.spacing`	vertical spacing between each plot. Can be single value or vector of length layout.height - 1
`xlab.label`	bottom x-axis main label
`xlab.cex`	bottom x-axis main label cex
`ylab.label`	left side y-axis label
`ylab.label.right`	right side y-axis label

`ylab.cex`	y-axis label cex
`main.cex`	main label cex
`legend`	legend for the plot
`left.padding`	padding from the left side of the frame
`ylab.axis.padding`	padding between axis and y label of plots. Can be single value or vector of length layout.width
`xlab.axis.padding`	padding between axis and x label of plots. Can be single value or vector of length layout.height
`bottom.padding`	padding from the bottom side of the frame
`top.padding`	padding from the top side of the frame
`right.padding`	padding from the right side of the frame
`layout.skip`	list specifiying locations to skip plots. Must be vector of length layout.width*layout.height
`left.legend.padding`	padding between legend and left side of figure (can use without a legend)
`right.legend.padding`	padding between legend and right side of figure (can use without a legend)
`bottom.legend.padding`	padding between legend and bottom side of figure (can use without a legend)
`top.legend.padding`	padding between legend and top side of figure (can use without a legend)
`description`	description of what plot is displaying
`size.units`	the units the height and width of file represent
`enable.warnings`	enables warnings to be output
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Jeff Green

Examples

```
set.seed(12345);
# begin by creating the individual plots which will be combined into a multiplot
dist <- data.frame(
    a = rnorm(100, 1),
    b = rnorm(100, 3),
    c = rnorm(100, 5)
    );

simple.data <- data.frame(
    x = c(dist$a, dist$b, dist$c),
    y = rep(LETTERS[1:3], each = 100)
    );
fill.squares <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), ncol = 3, byrow = TRUE);
rownames(fill.squares) <- c("Drug I only", "Drug II only" , "Drugs I & II");
colnames(fill.squares) <- levels(factor(simple.data$y));

# Create plot # 1
simple.boxplot <- create.boxplot(
    formula = x ~ y,
    data = simple.data,
    xaxis.lab = c('','',''),
    main.x = 0.57,
    ylab.label = 'Sugar Level',
    xlab.label = '',
    col = 'lightgrey',
    xaxis.tck = c(0,0),
    yaxis.tck = c(1,0),
    yaxis.lab = seq(-1,8,2) ,
    yat = seq(-1,8,2),
    left.padding = 0,
    right.padding = 0,
    lwd = 2
    );

# Create plot # 2
simple.heatmap <- create.heatmap(
    x = t(fill.squares),
    clustering.method = 'none',
    shrink = 0.8,
    yaxis.lab = c(3,2,3),
    yaxis.tck = 1,
    xaxis.lab = c('A','B','C'),
    ylab.label = 'Drug Regimen',
    xlab.label = 'Patient Group',
    colour.scheme = c("white", "grey20"),
    fill.colour = "white",
    print.colour.key = FALSE,
```

```
    left.padding = 0,
    xaxis.tck = c(1,0),
    right.padding = 0,
    xaxis.rot = 0
    );

create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_Simple', fileext = '.tiff'),
    plot.objects = list(simple.boxplot,simple.heatmap),
    y.spacing = 1,
    ylab.axis.padding = 2,
    main = 'Simple',
    top.padding = 2,
    resolution = 200
    );

# Create plot # 2
simple.heatmap.with.legends <- create.heatmap(
    x = t(fill.squares),
    shrink = 0.8,
    yaxis.lab = c(3,2,3),
    yaxis.tck = 1,
    xaxis.lab = c('A','B','C'),
    ylab.label = 'Drug Regimen',
    xlab.label = '',
    colour.scheme = c("white", "grey20"),
    fill.colour = "white",
    left.padding = 0,
    xaxis.tck = c(1,0),
    right.padding = 0,
    xaxis.rot = 0
    );

create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_Simple_Legends', fileext = '.tiff'),
    plot.objects = list(simple.boxplot,simple.heatmap.with.legends),
    y.spacing = 1,
    ylab.axis.padding = 2,
    main = 'Simple',
    top.padding = 2,
    resolution = 200
    );


# Create plot # 1
simple.boxplot2 <- create.boxplot(
    formula = x ~ y,
    data = simple.data,
    ylab.label = 'Sugar Level',
    xlab.label = '',
    col = 'lightgrey',
    xaxis.tck = c(0,0),
    xaxis.lab = c('','',''),
```

```
        yaxis.tck = c(1,0),
        yaxis.lab = seq(-1,8,2),
        yat = seq(-1,8,2),
        left.padding = 0,
        right.padding = 0,
        lwd = 2
        );

    simple.violin2 <- create.violinplot(
        formula = x ~ y,
        data = simple.data,
        col = 'lightgrey',
        yaxis.tck = c(0,0),
        xlab.label = '',
        ylab.label = '',
        yaxis.lab = NULL,
        xaxis.lab = c('','',''),
        xaxis.tck = c(0,0)
        );

    # Create plot # 2
    simple.heatmap2 <- create.heatmap(
        x = t(fill.squares),
        clustering.method = 'none',
        shrink = 0.8,
        yaxis.lab = c(1,2,3),
        yaxis.tck = 1,
        xaxis.lab = c('A','B','C'),
        ylab.label = 'Drug Regimen',
        colour.scheme = c("white", "grey20"),
        fill.colour = "white",
        print.colour.key = FALSE,
        left.padding = 0,
        xaxis.tck = c(3,0),
        right.padding = 0,
        xaxis.rot = 0,
        ylab.cex = 2
        );


    create.multipanelplot(
        # filename = tempfile(pattern = 'Multipanelplot_Simple_Layout', fileext = '.tiff'),
        plot.objects = list(simple.boxplot2,
        simple.violin2,simple.heatmap2),
        layout.width = 2,
        layout.height = 2,
        xlab.label = 'Patient Group',
        main = 'Simple Layout',
        top.padding = 2,
        plot.objects.heights = c(3,1),
        x.spacing = 1,
        y.spacing = 1
        );
```

```
all.data <- data.frame(
    a = rnorm(n = 25, mean = 0, sd = 0.75),
    b = rnorm(n = 25, mean = 0, sd = 0.75),
    c = rnorm(n = 25, mean = 0, sd = 0.75),
    d = rnorm(n = 25, mean = 0, sd = 0.75),
    e = rnorm(n = 25, mean = 0, sd = 0.75),
    f = rnorm(n = 25, mean = 0, sd = 0.75),
    x = rnorm(n = 25, mean = 5),
    y = seq(1, 25, 1)
    );
# create the plot -- this allows for previewing of the individual plot
barplot.formatted <- create.barplot(
    formula = x ~ y,
    data = all.data[,7:8],
    yaxis.tck = c(1,0),
    border.lwd = 0,
    col = 'grey',
    xlab.label = '',
    xat = c(-100),
    ylab.label = '',
    yaxis.lab = seq(1, ceiling(max(all.data$x)), 1),
    yat = seq(1, ceiling(max(all.data$x)), 1),
    yaxis.cex = 1.5
    );


heatmap.formatted <- create.heatmap(
    x = all.data[,1:6],
    clustering.method = 'none',
    colour.scheme = c('magenta','white','green'),
    print.colour.key = FALSE,
    xlab.label = '',
    yaxis.tck = c(1,0),
    xaxis.tck = c(1,0),
    xat = c(1:25),
    yaxis.lab = c("BRCA1", "BRCA2", "APC", "TIN", "ARG", "FOO"),
    yat = c(1,2,3,4,5,6),
    xaxis.lab = c(1:25),
    xaxis.rot = 0,
    yaxis.cex = 1.5
    );

create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_formatted', fileext = '.tiff'),
    plot.objects = list(barplot.formatted, heatmap.formatted),
    plot.objects.heights = c(1,3),
    y.spacing = -3.75,
    main = 'Formatted',
    top.padding = 0
    );

data.bars <- data.frame(
```

```
        x = sample(x = 5:35, size = 10),
        y = seq(1,10,1)
        );

    data.cov <- data.frame(
        x = rnorm(n = 10, mean = 0, sd = 0.75),
        y = rnorm(n = 10, mean = 0, sd = 0.75),
        z = rnorm(n = 10, mean = 0, sd = 0.75)
        );

    # Create main barplot
    bars <- create.barplot(
        formula = x~y,
        data = data.bars,
        ylimits = c(0,35),
        ylab.label = '',
        sample.order = 'increasing',
        border.lwd = 0,
        yaxis.lab = seq(5,35,5),
        yat = seq(5,35,5),
        yaxis.tck = c(0,0),
        xlab.label = ''
        );

    # Make covariate bars out of heatmaps
    cov.1 <- create.heatmap(
        x = as.matrix(data.bars$y),
        clustering.method = 'none',
        scale.data = FALSE,
        colour.scheme = default.colours(4),
        grid.col = TRUE,
        col.colour = 'black',
        # col.lwd = 10,
        total.col = 5,
        print.colour.key = FALSE,
        yaxis.tck = 0,
        axes.lwd = 0
        );

    cov.2 <- create.heatmap(
        x = as.matrix(data.cov$y),
        clustering.method = 'none',
        scale.data = FALSE,
        colour.scheme = c("lightblue","dodgerblue2", "dodgerblue4"),
        grid.col = TRUE,
        col.colour = 'black',
        # col.lwd = 10,
        total.col = 4,
        print.colour.key = FALSE,
        yaxis.tck = 0
        );

    cov.3 <- create.heatmap(
```

```
    x = as.matrix(data.cov$z),
    clustering.method = 'none',
    scale.data = FALSE,
    colour.scheme = c("grey","coral1"),
    grid.col = TRUE,
    col.colour = 'black',
    # col.lwd = 10,
    total.col = 3,
    print.colour.key = FALSE,
    yaxis.tck = 0
    );


legendG <- legend.grob(
    list(
        legend = list(
            colours = default.colours(4),
            title = "Batch",
            labels = LETTERS[1:4],
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            ),
        legend = list(
            colours = c("lightblue","dodgerblue2","dodgerblue4"),
            title = "Grade",
            labels = c("Low","Normal","High"),
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            ),
        legend = list(
            colours = c("grey","coral1"),
            title = "Biomarker",
            labels = c("Not present","Present"),
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            )
        ),
        label.cex = 1.25,
     title.cex = 1.25,
     title.just = 'left',
    title.fontface = 'bold.italic',
    size = 3,
    layout = c(1,3)
     );

create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_Barchart', fileext = '.tiff'),
```

```
    plot.objects = list(bars, cov.3, cov.2, cov.1 ),
    plot.objects.heights = c(1, 0.1,0.1,0.1),
    legend = list(right = list(fun = legendG)),
    ylab.label = 'Response to Treatment',
    main = 'Bar Chart',
    x.spacing = 0,
    y.spacing = 0.1
    );

# Set up plots for complex example

# Dotmap
spot.sizes <- function(x) { 0.5 * abs(x); }
dotmap.dot.colours <- c('red','blue');
spot.colours <- function(x) {
    colours <- rep('white', length(x));
    colours[sign(x) == -1] <- dotmap.dot.colours[1];
    colours[sign(x) ==  1] <- dotmap.dot.colours[2];
    return(colours);
    };

# Dotmap colours
orange <- rgb(249/255, 179/255, 142/255);
blue <- rgb(154/255, 163/255, 242/255);
green <- rgb(177/255, 213/255, 181/255);
bg.colours <- c(green, orange, blue, 'gold', 'skyblue', 'plum');

dotmap <- create.dotmap(
    x = CNA[1:15,1:58],
    bg.data = SNV[1:15,1:58],
    # Set the colour scheme
    colour.scheme = bg.colours,
    # Set the breakpoints for the colour scheme (determined from the data)
    at = c(0,1,2,4,6,7,8),
    # Specify the total number of colours (+1 for the fill colour)
    total.colours = 7,
    col.colour = 'white',
    row.colour = 'white',
    bg.alpha = 1,
    yaxis.tck = c(1,0),
    fill.colour = 'grey95',
    spot.size.function = spot.sizes,
    spot.colour.function = spot.colours,
    xaxis.tck = 0,
    xaxis.lab = c(rep('',100)),
    bottom.padding = 0,
    top.padding = 0,
    left.padding = 0,
    right.padding = 0,
    yaxis.cex = 1
    );

# Dotmap legend
```

```
dotmap.legend <- list(
    legend = list(
        colours = bg.colours,
        labels = c('Nonsynonymous','Stop Gain','Frameshift deletion',
            'Nonframeshift deletion', 'Splicing', 'Unknown'),
        border = 'white',
        title = 'SNV',
        pch = 15
        ),
    legend = list(
        colours = dotmap.dot.colours,
        labels = c('Gain','Loss'),
        border = 'white',
        title = 'CNA',
        pch = 19
        )
    );

dotmap.legend.grob <- legend.grob(
    legends = dotmap.legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Covariates
cov.colours <- c(
    c('dodgerblue','pink'),
    c('grey','darkseagreen1','seagreen2','springgreen3','springgreen4'),
    c('peachpuff','tan4')
    );

# the heatmap expects numeric data
cov.data <- patient[-c(4:9)];
cov.data[cov.data == 'male'] <- 1;
cov.data[cov.data == 'female'] <- 2;
cov.data[is.na(cov.data)] <- 3;
cov.data[cov.data == 'I'] <- 4;
cov.data[cov.data == 'II'] <- 5;
cov.data[cov.data == 'III'] <- 6;
cov.data[cov.data == 'IV'] <- 7;
cov.data[cov.data == 'MSS'] <- 8;
cov.data[cov.data == 'MSI-High'] <- 9;
cov.data$sex <- as.numeric(cov.data$sex);
cov.data$stage <- as.numeric(cov.data$stage);
cov.data$msi <- as.numeric(cov.data$msi);

covariates <- create.heatmap(
    x = cov.data,
    clustering.method = 'none',
    colour.scheme = as.vector(cov.colours),
    total.colours = 10,
    row.colour = 'white',
```

```
    col.colour = 'white',
    grid.row = TRUE,
    grid.col = TRUE,
    xaxis.lab = c(rep('',100)),
    yaxis.lab = c('Sex','Stage','MSI'),
    yaxis.tck = c(0,0),
    xaxis.tck = c(0,0),
    xat = c(1:100),
    print.colour.key = FALSE,
    yaxis.cex = 1,
    bottom.padding = 0,
    top.padding = 0,
    left.padding = 0,
    right.padding = 0
    );

## Warning: number of columns exceeded limit (50), column lines are
## turned off. Please set "force.grid.col" to TRUE to override this

# Coviate Legends
cov.legends <- list(
    legend = list(
        colours = cov.colours[8:9],
        labels = c('MSS','MSI-High'),
        border = 'white',
        title = 'MSI'
        ),
    legend = list(
        colours = cov.colours[3:7],
        labels = c('NA', 'I','II','III','IV'),
        border = 'white',
        title = 'Stage'
        ),
    legend = list(
        colours = cov.colours[1:2],
        labels = c('Male','Female'),
        border = 'white',
        title = 'Sex'
        )
    );

cov.legend.grob <- legend.grob(
    legends = cov.legends,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7,
    layout = c(3,1)
    );


create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_with_heatmap', fileext = '.tiff'),
    plot.objects = list(dotmap,covariates),
```

```
    plot.objects.heights = c(1,0.2),
    y.spacing = -0.8,
    main = 'Dotmap',
    top.padding = 2,
    layout.height = 2,
    legend = list(
        bottom = list(
            x = 0.10,
            y = 0.50,
            fun = cov.legend.grob
            ),
        right = list(
            x = 0.10,
            y = 0.50,
            fun = dotmap.legend.grob
            )
        ),
    resolution = 300
    );

# Add more plots, using more complex layout
# grouped barplot
groupedbar.colours <- c('indianred1','indianred4');

count.SNV <- apply(SNV[1:15,], 2, function(x){length(which(!is.na(x)))});
count.CNA <- apply(CNA[1:15,], 2, function(x){length(which(!(x==0)))});

grouped.data <- data.frame(
    values = c(count.SNV, count.CNA),
    samples = rep(colnames(SNV),2),
    group = rep(c('SNV','CNA'), each = 58)
    );

grouped.barplot <- create.barplot(
    formula = values ~ samples,
    data = grouped.data,
    groups = grouped.data$group,
    col = groupedbar.colours,
    top.padding = 0,
    bottom.padding = 0,
    left.padding = 0,
    right.padding = 0,
    border.col = 'white',
    xlab.label = '',
    ylab.label = 'Mutation',
    yaxis.lab = c(0,5,10,15),
    yat = c(0,5,10,15),
    xaxis.lab = c(rep('',100)),
    yaxis.tck = c(0,0),
    xaxis.tck = c(0,0),
    ylab.cex = 1.5,
    yaxis.cex = 1,
    axes.lwd = 2
```

```
    );

# stacked barplot
col.one <- rgb(255/255, 225/255, 238/255);
col.two <- rgb(244/255, 224/255, 166/255);
col.thr <- rgb(177/255, 211/255, 154/255);
col.fou <- rgb(101/255, 180/255, 162/255);
col.fiv <- rgb(51/255, 106/255, 144/255);
stackedbar.colours <- c(col.one, col.two, col.thr, col.fou, col.fiv, 'orchid4');
stacked.data.labels <- c('C>A/G>T','C>T/G>A','C>G/G>C','T>A/A>T','T>G/A>C', 'T>C/A>G');

stacked.data <- data.frame(
   values = c(patient$prop.CAGT, patient$prop.CTGA, patient$prop.CGGC, patient$prop.TAAT,
        patient$prop.TGAC, patient$prop.TCAG),
    divisions = rep(rownames(patient), 6),
    group = rep(stacked.data.labels, each = 58)
    );

# Generate stacked barplot
stacked.barplot <- create.barplot(
    formula = values ~ divisions,
    data = stacked.data,
    groups = stacked.data$group,
    stack = TRUE,
    col = stackedbar.colours,
    border.col = 'white',
    main = '',
    xlab.label = '',
    ylab.label = 'Proportion',
    yaxis.lab = c(0,0.4,0.8),
    yat = c(0,0.4,0.8),
    xaxis.lab = c(rep('',100)),
yaxis.tck = c(0,0),
    xaxis.tck = c(0,0),
    ylab.cex = 1.5,
    yaxis.cex = 1,
    axes.lwd = 2
    );

# barchart legends
stackedbar.legend <- list(
    legend = list(
        colours = rev(stackedbar.colours),
        labels = rev(stacked.data.labels),
        border = 'white'
        )
    );

groupedbar.legend <- list(
    legend = list(
        colours = groupedbar.colours,
        labels = c('CNA','SNV'),
        border = 'white'
```

```
        )
    );

groupedbar.legend.grob <- legend.grob(
    legends = groupedbar.legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

stackedbar.legend.grob <- legend.grob(
    legends = stackedbar.legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Expression change Segplot
# locate matching genes
rows.to.keep <- which(match(rownames(microarray), rownames(SNV)[1:15], nomatch = 0) > 0);

segplot.data <- data.frame(
    min = apply(microarray[rows.to.keep,1:58], 1, min),
    max = apply(microarray[rows.to.keep,1:58], 1, max),
    median = apply(microarray[rows.to.keep,1:58], 1, median),
    order = seq(1,15,1)
    );

segplot <- create.segplot(
    formula = order ~ min + max,
    data = segplot.data,
    main = '',
    xlab.label = '',
    ylab.label = '',
    centers = segplot.data$median,
    yaxis.lab = c('','','','','',''),
    xaxis.lab = c('0','2','4','6','8'),
    xat = c(0,2,4,6,8),
    yaxis.tck = c(0,0),
    xaxis.tck = c(1,0),
    axes.lwd = 2,

    top.padding = 0,
    left.padding = 0,
    right.padding = 0,
    bottom.padding = 0
    );
# Create multiplot

plots <- list(grouped.barplot,stacked.barplot,dotmap, segplot,covariates);
create.multipanelplot(
    main.x = 0.47,
    main.y = 0.5,
```

```
    plot.objects = plots,
    plot.objects.heights = c(0.3, 0.3, 1, 0.15),
    plot.objects.widths = c(1,0.2),
    # filename = tempfile(pattern = 'Multipanelplot_Complex', fileext = '.tiff'),
    layout.height = 4,
    layout.width = 2,
    x.spacing = 0.2,
    left.padding = 0,
    layout.skip = c(FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,FALSE,TRUE),
    y.spacing = c(-1.35,-1.35,-1.5),
    ylab.axis.padding = c(1,0),
    legend = list(
        left = list(
            fun = dotmap.legend.grob,
            args = list(
                key = list(
                    points = list(
                        pch = c(15,15,19,19)
                        )
                    )
                )
            )
    ),
    height = 12,
    width = 12,
    main = 'Complex',
    top.padding = 2
    );
# Create a multiplot with a heatmap, key like legend and barplot

# First create a heatmap object
simple.heatmap <- create.heatmap(patient[, 4:6],
   clustering.method = 'none',
   print.colour.key = FALSE,
   same.as.matrix = FALSE,
   colour.scheme = c('gray0','grey100'),
   fill.colour = 'grey95',
   xaxis.lab = c(rep('',100)),
   xat = c(0,1,2,3,4,5,6,7,8),
   yaxis.lab = c('','',''),
   yat = c(0,1,2),
   xlab.label = ''
);


# and a simple bar plot
pvals <- data.frame(
    order = c(1:3),
    pvalue = -log10(c(0.0004, 0.045, 0.0001)),
    stringsAsFactors = FALSE
        )
#create bar plot
simple.bar <- create.barplot(
```

```
    formula = order ~ rev(pvalue),
    data = pvals,
    xlimits = c(0,5),
    plot.horizontal=TRUE,
    xlab.label = '',
    ylab.label = '',
    yaxis.lab = c(1,2,3)
    );

# then the covariates heatmap
cov.colours <- c(
    c('dodgerblue','pink'),
    c('grey','darkseagreen1','seagreen2','springgreen3','springgreen4'),
    c('peachpuff','tan4')
    );

# the heatmap expects numeric data
cov.data <- patient[-c(4:9)];
cov.data[cov.data == 'male'] <- 1;
cov.data[cov.data == 'female'] <- 2;
cov.data[is.na(cov.data)] <- 3;
cov.data[cov.data == 'I'] <- 4;
cov.data[cov.data == 'II'] <- 5;
cov.data[cov.data == 'III'] <- 6;
cov.data[cov.data == 'IV'] <- 7;
cov.data[cov.data == 'MSS'] <- 8;
cov.data[cov.data == 'MSI-High'] <- 9;
cov.data$sex <- as.numeric(cov.data$sex);
cov.data$stage <- as.numeric(cov.data$stage);
cov.data$msi <- as.numeric(cov.data$msi);

covariates <- create.heatmap(
    x = cov.data,
    clustering.method = 'none',
    colour.scheme = as.vector(cov.colours),
    total.colours = 10,
    row.colour = 'white',
    col.colour = 'white',
    grid.row = TRUE,
    grid.col = TRUE,
    yaxis.tck = 0,
    print.colour.key = FALSE,
    xaxis.lab = c('','',''),
    xlab.label = '',
    xat = c(1,2,3)
    );

## Warning: number of columns exceeded limit (50), column
## lines are turned off. Please set "force.grid.col" to TRUE to override this

covariates2 <- create.heatmap(
    x = patient[4],
    clustering.method = 'none',
```

```
    colour.scheme = c("#00007F", "#007FFF"),
    row.colour = 'white',
    col.colour = 'white',
    grid.row = TRUE,
    grid.col = TRUE,
    yaxis.tck = 0,
    print.colour.key = FALSE,
    xaxis.lab = c('','',''),
    xlab.label = '',
    xat = c(1,2,3)
    );

## Warning: number of rows exceeded limit (50), row
## lines are turned off. Please set "force.grid.row" to TRUE to override this


cov.legends <- list(
    legend = list(
        colours = c("white", "black"),
        labels = c('0','2'),
        border = 'grey',
        title = 'Tumour Mass (kg)',
        continuous = TRUE,
        height = 3
        ),
    legend = list(
        colours = cov.colours[8:9],
        labels = c('MSS','MSI-High'),
        border = 'white',
        title = 'MSI'
        ),
    legend = list(
        colours = cov.colours[3:7],
        labels = c('NA', 'I','II','III','IV'),
        border = 'white',
        title = 'Stage'
        ),
    legend = list(
        colours = cov.colours[1:2],
        labels = c('Male','Female'),
        border = 'white',
        title = 'Sex'
        ),
    legend = list(
        colours = c("#00007F", "#007FFF"),
        labels = c('0.09','0.72'),
        border = 'grey',
        title = 'CAGT',
        continuous = TRUE,
        height = 2,
        width = 3,
        angle = -90,
        tck = 1,
```

```
        tck.number = 2,
        at = c(0,100)
        )
    );

cov.legend.grob <- legend.grob(
    legends = cov.legends,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Now bring it was together using multiplot
create.multipanelplot(
   # filename = tempfile(pattern = 'Multipanelplot_continousLegend', fileext = '.tiff'),
    plot.objects = list(simple.heatmap, simple.bar,covariates2,covariates),
    plot.objects.heights = c(1,0.1,0.35),
    plot.objects.widths = c(1,0.25),
    layout.height = 3,
    layout.width = 2,
    layout.skip = c(FALSE, FALSE,FALSE,TRUE,FALSE,TRUE),
    y.spacing = -0.1,
    x.spacing = 0.5,
    legend = list(
        left = list(
            fun = cov.legend.grob
            )
        ),
    main = 'Continous Legend',
    top.legend.padding = 4,
    top.padding = -2,
    left.padding = 1
    # This parameter must be set for the legend to appear
    );

 create.multipanelplot(
    # filename = tempfile(pattern = 'Multipanelplot_manyPlots', fileext = '.tiff'),
    main = 'Large Scale',
    plot.objects = list(
simple.boxplot,
simple.heatmap,
simple.bar,
barplot.formatted,
dotmap,
grouped.barplot,
stacked.barplot,
covariates,
covariates2,
heatmap.formatted
),
    plot.objects.heights = c(1,1,1,1),
    plot.objects.widths = c(1,1, 1,1),
    layout.height = 4,
```

```
    layout.width = 4,
    top.legend.padding = 3,
    layout.skip = c(FALSE, FALSE,FALSE,FALSE,FALSE,TRUE,
TRUE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,TRUE,TRUE),
    y.spacing = c(-1,-1,-1),
    x.spacing = c(1,2,3),
    legend = list(
        left = list(
            fun = cov.legend.grob
            )
        ),
    height = 12,
    width = 12
    # This parameter must be set for the legend to appear
    );
```

`create.multiplot` *Joins plots together*

Description

Merges together multiple plots in the specified layout

Usage

```
create.multiplot(
plot.objects,
filename = NULL,
panel.heights = c(1,1),
panel.widths = 1,
main = NULL,
main.just = "center",
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
main.key.padding = 1,
ylab.padding = 5,
xlab.padding = 5,
xlab.to.xaxis.padding = 2,
right.padding = 1,
left.padding = 1,
top.padding = 0.5,
bottom.padding = 0.5,
xlab.label = NULL,
ylab.label = NULL,
xlab.cex = 2,
```

```
ylab.cex = 2,
xlab.top.label = NULL,
xaxis.top.tck.lab = NULL,
xat.top = TRUE,
xlab.top.cex = 2,
xaxis.top.idx = NULL,
xlab.top.col = 'black',
xlab.top.just = "center",
xlab.top.x = 0.5,
xlab.top.y = 0,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.labels = TRUE,
yaxis.labels = TRUE,
xaxis.alternating = 1,
yaxis.alternating = 1,
xat = TRUE,
yat = TRUE,
xlimits = NULL,
ylimits = NULL,
xaxis.rot = 0,
xaxis.rot.top = 0,
xaxis.fontface = 'bold',
y.tck.dist=0.5,
x.tck.dist=0.5,
yaxis.fontface = 'bold',
x.spacing = 1,
y.spacing = 1,
x.relation = 'same',
y.relation = 'same',
xaxis.tck = c(0.75,0.75),
yaxis.tck = c(0.75,0.75),
axes.lwd = 1.5,
key.right.padding = 1,
key.left.padding = 1,
key.bottom.padding = 1,
xlab.key.padding = 0.5,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
key = list(text = list(lab = c(''))),
legend =  NULL,
print.new.legend = FALSE,
merge.legends = FALSE,
plot.layout = c(1,length(plot.objects)),
layout.skip=rep(FALSE,length(plot.objects)),
```

```
description = 'Created with BoutrosLab.plotting.general',
plot.labels.to.retrieve = NULL,
style = 'BoutrosLab',
remove.all.border.lines = FALSE,
preload.default = 'custom',
plot.for.carry.over.when.same = 1,
get.dendrogram.from = NULL,
dendrogram.right.size = NULL,
dendrogram.right.x = NULL,
dendrogram.right.y = NULL,
        dendrogram.top.size = NULL,
        dendrogram.top.x = NULL,
        dendrogram.top.y = NULL,
        use.legacy.settings = FALSE
);
```

Arguments

`plot.objects`	A list of plot objects. Goes in this order: Bottom Left, Bottom Right, Top Left, Top Right
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`panel.heights`	A vector specifying relative heights of the panels. Default is c(1,1)
`panel.widths`	A vector specifying relative widths of the panels. Default is 1
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 3
`main.key.padding`	A number specifying the distance of main to plot, defaults to 1
`ylab.padding`	A number specifying the distance of y-axis to plot, defaults to 5
`xlab.padding`	A number specifying the distance of x-axis to plot, defaults to 5
`xlab.to.xaxis.padding`	A number specifying the distance between xaxis and xlabel, defaults to 2
`right.padding`	A number specifying the distance to the right margin, defaults to 1
`left.padding`	A number specifying the distance to the left margin, defaults to 1
`top.padding`	A number specifying the distance to the top margin, defaults to 0.5
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.5
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis labels, defaults to 1.5
`ylab.cex`	Size of y-axis labels, defaults to 1.5

`xlab.top.label`	The label for the top x-axis
`xaxis.top.tck.lab`	A vector of tick labels for the top x-axis. Currently only supports labelling a single top x-axis in the plot
`xat.top`	A vector specifying tick positions for the top x-axis. Currently only supports a single top x-axis in the plot. Note when labelling a top x-axis even if you're not labelling a bottom x-axis labels xat must still be defined (eg as a list of empty vectors) or it will lead to unpredictable labelling
`xlab.top.cex`	Size of top x-axis label
`xaxis.top.idx`	Index of the plot for which you want top x-axis tick labels. Defaults to the last plot specified. Currently only supports one plot.
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2
`xaxis.labels`	Names to give the x-axis labels, defaults to lattice default behaviour
`yaxis.labels`	Names to give the y-axis labels, defaults to lattice default behaviour
`xaxis.alternating`	Gives control of axis tick marks (1 bottom only, 2 top only, 3 both top and bottom), default to 1 which means only bottom axis tick marks are drawn, set to 0 to remove tick marks
`yaxis.alternating`	Gives control of axis labelling, defaults to 1 which means only left axis labels are drawn, set to 0 to remove tick marks
`xat`	Vector listing where the x-axis labels should be drawn
`yat`	Vector listing where the y-axis labels should be drawn
`xlimits`	Vector listing where the x-axis limits should be for each subplot. Defaults to NULL to let R figure out the limits
`ylimits`	Vector listing where the y-axis limits should be for each subplot. Defaults to NULL to let R figure out the limits
`xaxis.rot`	Rotation of bottom x-axis labels
`xaxis.rot.top`	Rotation of top x-axis labels
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`x.spacing`	A number specifying the horizontal distance between plots, defaults to 1
`y.spacing`	A number specifying the vertical distance between plots, defaults to 1
`x.relation`	A character string that determines how x-axis limits are calculated for each panel. Possible values are "same" (default), "free" and "sliced". See ?xyplot

`y.relation`	A character string that determines how y-axis limits are calculated for each panel. Possible values are “same” (default), “free” and “sliced”. See ?xyplot
`xaxis.tck`	A vector of length 2 that determines the size of x-axis tick marks. Defaults to c(0.75, 0.75).
`yaxis.tck`	A vector of length 2 that determines the size of y-axis tick marks. Defaults to c(0.75, 0.75).
`x.tck.dist`	A number specifying the distance between x-axis labels and tick marks. Defaults to 0.5.
`y.tck.dist`	A number specifying the distance between y-axis labels and tick marks. Defaults to 0.5.
`axes.lwd`	Width of border. Note it also changes the colourkey border and ticks
`key.right.padding`	Space between right-most plot and any keys/legends
`key.left.padding`	Space between left-most plot and any keys/legends
`key.bottom.padding`	Space between bottom-most plot and any keys/legends
`xlab.key.padding`	Space between bottom-most xlab and any keys/legends
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`key`	Add a key to the plot: see xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See ?xyplot.
`print.new.legend`	Override default behaviour of merging legends imported from plots, can specify custom legend, default is FALSE. TRUE will cancel merge.legends functionality
`merge.legends`	FALSE means only legend from first plot is used, TRUE retrieves legends from all plots. Multiple legends share the same “space”:see c.trellis.
`plot.layout`	A vector specifying the layout of the plots, defaults to a single column/ c(1,length(plot.objects))
`layout.skip`	A vector specifying which positions in the layout grid to leave blank/skip, defaults to not skipping any spots in the layout / rep(FALSE,length(plot.objects)). Goes in this order: Bottom Left, Bottom Right, Top Left, Top Right
`description`	Short description of image/plot; default NULL.
`plot.labels.to.retrieve`	a vector of the indices referencing which plots in plot.objects should have there limits, at, and axis labels retrived in the multiplot vs using the arguments specified to multiplot

`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`remove.all.border.lines`	defaults to FALSE. Flag for whether all borders around plots should be removed.
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`plot.for.carry.over.when.same`	which plot
`get.dendrogram.from`	which plot to retrieve dendrogram from
`dendrogram.right.size`	size of right side dendrogram
`dendrogram.right.x`	x position of right side dendrogram
`dendrogram.right.y`	y position of right side dendrogram
`dendrogram.top.size`	size of top side dendrogram
`dendrogram.top.x`	x position of top side dendrogram
`dendrogram.top.y`	y position of top side dendrogram
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Ken Chu and Denise Mak

Examples

```
set.seed(12345);

# begin by creating the individual plots which will be combined into a multiplot
dist <- data.frame(
    a = rnorm(100, 1),
    b = rnorm(100, 3),
    c = rnorm(100, 5)
    );

simple.data <- data.frame(
    x = c(dist$a, dist$b, dist$c),
    y = rep(LETTERS[1:3], each = 100)
    );

fill.squares <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), ncol = 3, byrow = TRUE);
rownames(fill.squares) <- c("Drug I only", "Drug II only" , "Drugs I & II");
colnames(fill.squares) <- levels(factor(simple.data$y));

# Create plot # 1
simple.boxplot <- create.boxplot(
    formula = x ~ y,
    data = simple.data,
    col = 'lightgrey'
    );

# Create plot # 2
simple.heatmap <- create.heatmap(
        x = t(fill.squares),
        clustering.method = 'none',
        shrink = 0.8,
        colour.scheme = c("white", "grey20"),
        fill.colour = "white",
        print.colour.key = FALSE
        );

# Simple example of multiplot
# This example uses the defaults set in simple.heatmap and simple.boxplot
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Simple', fileext = '.tiff'),
        plot.objects = list(simple.heatmap, simple.boxplot),
        main = "Simple",
        xlab.label = c("Patient Group"),
        # The plotting function throws an error if this is not included
        ylab.label = c("Sugar Level", "Drug Regimen"),
        ylab.padding = 7,
        # Parameters set in the multiplot will override settings in individual plots
        xaxis.cex = 0.7,
        yaxis.cex = 0.7,
resolution = 100
        );
```

```
# Simple example of multiplot with adjusted plot sizes
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Simple_Plot_Sizes', fileext = '.tiff'),
        plot.objects = list(simple.heatmap, simple.boxplot),
        main = "Simple plot sizes",
        xlab.label = c("Patient Group"),
        # y-axis labels must be spaced with tabs or spaces to properly align
        ylab.label = c("", "Sugar Level", "", "Drug Regimen"),
        ylab.padding = 7,
        xaxis.cex = 0.7,
        yaxis.cex = 0.7,
        # Set the relative heights of the plots
        panel.heights = c(3,1),
resolution = 100
        );

simple.violin <- create.violinplot(
    formula = x ~ y,
    data = simple.data,
    col = 'lightgrey'
    );

# Simple example of multiplot with custom layout
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Simple_Layout', fileext = '.tiff'),
    plot.objects = list(simple.heatmap, simple.boxplot, simple.violin),
    main = "Simple layout",
    xlab.label = c("Patient Group"),
    ylab.label = c("", "Sugar Level", "", "Drug Regimen"),
    ylab.padding = 7,
    xaxis.cex = 0.7,
    yaxis.cex = 0.7,
    panel.heights = c(3,1),
    # Set how many rows & columns are in the layout
    plot.layout = c(2,2),
    # Set whether to plot or not in the space (fills from bottom left to top right)
    layout.skip = c(FALSE, TRUE, FALSE, FALSE),
    # Move plots closer together
    x.spacing = 0,
    # Remove doubled internal axis
    yat = list(
        seq(1,3,1),
        seq(-2, 8, 2),
        c()
        ),
    resolution = 100
    );

# Example of how to take parameter values from individual plots
# This programming structure allows for including the individual customization
# of plots to the final multiplot
all_data <- data.frame(
    a = rnorm(n = 25, mean = 0, sd = 0.75),
```

```
        b = rnorm(n = 25, mean = 0, sd = 0.75),
        c = rnorm(n = 25, mean = 0, sd = 0.75),
        d = rnorm(n = 25, mean = 0, sd = 0.75),
        e = rnorm(n = 25, mean = 0, sd = 0.75),
        f = rnorm(n = 25, mean = 0, sd = 0.75),
        x = rnorm(n = 25, mean = 5),
        y = seq(1, 25, 1)
        );

    plot.heatmap <- function(all_data){
        # save the parameter values that will be reused in the multiplot
        multiplot_visuals <- list(
            xlab.label = '',
            xaxis.labels = NULL,
            xat = NULL,
            ylab.label = 'Genes of Interest',
            yaxis.labels = c("BRCA1", "BRCA2", "APC", "TIN", "ARG", "FOO"),
            yat = c(1,2,3,4,5,6)
            );

        # create the plot -- this allows for previewing of the individual plot
        heatmap.formatted <- create.heatmap(
            x = all_data[,1:6],
            clustering.method = 'none',
            colour.scheme = c('magenta','white','green'),
            print.colour.key = FALSE,
            xlab.label = multiplot_visuals$xlab.label,
            xaxis.lab = multiplot_visuals$xaxis.labels,
            xat = multiplot_visuals$xat,
            ylab.label = multiplot_visuals$ylab.label,
            yaxis.lab = multiplot_visuals$yaxis.labels,
            yat = multiplot_visuals$yat
            );

        # return both the plot and the relevant parameter values
        return(
            list(
                the_plot = heatmap.formatted,
                visuals = multiplot_visuals
                )
            )
    }

    plot.barplot <- function(all_data) {

        # save the parameter values that will be reused in the multiplot
        multiplot_visuals <- list(
            xlab.label = '',
            xaxis.labels = NULL,
            xat = NULL,
            ylab.label = 'Importance',
            yaxis.labels = seq(1, ceiling(max(all_data$x)), 1),
            yat = seq(1, ceiling(max(all_data$x)), 1)
```

```
        );

    # create the plot -- this allows for previewing of the individual plot
    barplot.formatted <- create.barplot(
        formula = x ~ y,
        data = all_data[,7:8],
        border.lwd = 0,
        col = 'grey',
        xlab.label = multiplot_visuals$xlab.label,
        xaxis.lab = multiplot_visuals$xaxis.labels,
        xat = multiplot_visuals$xat,
        ylab.label = multiplot_visuals$ylab.label,
        yaxis.lab = multiplot_visuals$yaxis.labels,
        yat = multiplot_visuals$yat
        );

    # return both the plot and the relevant parameter values
    return(
        list(
            the_plot = barplot.formatted,
            visuals = multiplot_visuals
            )
        )
    }

plot_functions <- c('plot.heatmap', 'plot.barplot');

# run the functions
all_plots <- lapply(
    plot_functions,
    function(funName){
        eval(parse(text = paste0(funName, '(all_data)')))
        }
    );

create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Formatting', fileext = '.tiff'),
    main = "Formatting",
    plot.objects = lapply(all_plots, function(aPlot) aPlot$the_plot),
    panel.heights = c(1,3),
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylab.padding = 8,
    yat = lapply(all_plots,function(aPlot) aPlot$visuals$yat),
    xlab.label = lapply(all_plots,function(aPlot) aPlot$visuals$xlab.label),
    ylab.label = rev(lapply(all_plots,function(aPlot) aPlot$visuals$ylab.label)),
    yaxis.labels = lapply(all_plots,function(aPlot) aPlot$visuals$yaxis.labels),
    resolution = 100
    );

data_bars <- data.frame(
    x = sample(x = 5:35, size = 10),
    y = seq(1,10,1)
```

```
    );

data_cov <- data.frame(
    x = rnorm(n = 10, mean = 0, sd = 0.75),
    y = rnorm(n = 10, mean = 0, sd = 0.75),
    z = rnorm(n = 10, mean = 0, sd = 0.75)
    );

# Create main barplot
bars <- create.barplot(
    formula = x~y,
    data = data_bars,
    ylimits = c(0,35),
    sample.order = 'increasing',
    border.lwd = 0
    );

# Make covariate bars out of heatmaps
cov_1 <- create.heatmap(
    x = as.matrix(data_bars$y),
    clustering.method = 'none',
    scale.data = FALSE,
    colour.scheme = default.colours(4),
    grid.col = TRUE,
    col.colour = 'black',
    # col.lwd = 10,
    total.col = 5,
    print.colour.key = FALSE,
    yaxis.tck = 0,
    axes.lwd = 0
    );

cov_2 <- create.heatmap(
    x = as.matrix(data_cov$y),
    clustering.method = 'none',
    scale.data = FALSE,
    colour.scheme = c("lightblue","dodgerblue2", "dodgerblue4"),
    grid.col = TRUE,
    col.colour = 'black',
    # col.lwd = 10,
    total.col = 4,
    print.colour.key = FALSE,
    yaxis.tck = 0
    );

cov_3 <- create.heatmap(
    x = as.matrix(data_cov$z),
    clustering.method = 'none',
    scale.data = FALSE,
    colour.scheme = c("grey","coral1"),
    grid.col = TRUE,
    col.colour = 'black',
    # col.lwd = 10,
```

```
    total.col = 3,
    print.colour.key = FALSE,
    yaxis.tck = 0
    );

# Generate legends outside of individual functions
legend <- legend.grob(
    list(
        legend = list(
            colours = default.colours(4),
            title = "Batch",
            labels = LETTERS[1:4],
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            ),
        legend = list(
            colours = c("lightblue","dodgerblue2","dodgerblue4"),
            title = "Grade",
            labels = c("Low","Normal","High"),
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            ),
        legend = list(
            colours = c("grey","coral1"),
            title = "Biomarker",
            labels = c("Not present","Present"),
            size = 3,
            title.cex = 1,
            label.cex = 1,
            border = 'black'
            )
        ),
    title.just = 'left'
    );

# Assemble plot using multiplot function
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Barchart', fileext = '.tiff'),
    main = 'Multiplot with bar chart',
    plot.objects = list(cov_3, cov_2, cov_1, bars),
    ylab.label = c("\t", "Response to treatment","\t"),
    xlab.label = "Sample characteristics",
    panel.heights = c(1, 0.05,0.05,0.05),
    y.spacing = c(-1, -1, -1, 0),
    xaxis.lab = NULL,
    yaxis.lab = list(NULL, NULL, NULL, seq(0, 350, 50)),
    legend = list(right = list(fun = legend)),
    print.new.legend = TRUE,
    xaxis.alternating = 0,
```

```
    main.cex = 1,
    ylab.cex = 1,
    xlab.cex = 1,
    xlab.to.xaxis.padding = -2,
    yaxis.cex = 1,
    description = "Multiplot example created by BoutrosLab.plotting.general",
    resolution = 200
    );


gene_data <- data.frame(
    x = rnorm(n = 25, mean = 0, sd = 0.75),
    y = rnorm(n = 25, mean = 0, sd = 0.75),
    z = rnorm(n = 25, mean = 0, sd = 0.75),
    v = rnorm(n = 25, mean = 0, sd = 0.75),
    w = rnorm(n = 25, mean = 0, sd = 0.75),
    a = rnorm(n = 25, mean = 0, sd = 0.75),
    b = rnorm(n = 25, mean = 0, sd = 0.75),
    c = rnorm(n = 25, mean = 0, sd = 0.75)
    );

# main heatmap
main <- create.heatmap(
    x = gene_data,
    xaxis.tck = 0,
    yaxis.tck = 0,
    colourkey.cex = 1,
    clustering.method = 'none',
    axes.lwd = 1,
    ylab.label = 'y',
    xlab.label = 'x',
    yaxis.fontface = 1,
    xaxis.fontface = 1,
    xlab.cex = 1,
    ylab.cex = 1,
    main.cex = 1,
    colour.scheme = c('red','white','turquoise')
    );

key_data <- data.frame(
    x <- seq(-50,50,1)
    );

# colour key for heatmap
key <- create.heatmap(
    x = key_data,
    clustering.method = 'none',
    scale.data = FALSE,
    colour.scheme = c('turquoise','white','red'),
    print.colour.key = FALSE,
    yaxis.tck = 0,
    xat = c(10,90),
    xaxis.lab = c('low', 'high')
```

```
    );

top_data <- data.frame(
    x = rnorm(n = 25, mean = 0, sd = 0.75),
    y = seq(1,25,1)
    );

# top barplot
top <- create.barplot(
    formula = x~y,
    data = top_data,
    border.lwd = 0
    );

side_data <- data.frame(
    x = rnorm(n = 8, mean = 0, sd = 0.75),
    y = seq(1,8,1)
    );

# side barplot
side <- create.barplot(
    formula = x~y,
    data = side_data,
    border.lwd = 0,
    sample.order = 'decreasing',
    plot.horizontal = TRUE
    );

# assembling final figure
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_with_heatmap', fileext = '.tiff'),
    main = 'Multiplot with heatmap',
    plot.objects = list(key, main, side, top),
    panel.heights = c(0.25, 1, 0.05),
    panel.widths = c(1, 0.25),
    plot.layout = c(2, 3),
    layout.skip = c(FALSE, TRUE, FALSE, FALSE, FALSE, FALSE),
    xaxis.alternating = 0,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1,
    ylab.cex = 1,
    xlab.label = c('\t', 'Samples', '\t', '    Importance'),
    ylab.label = c( 'Amount (g)', '\t', '\t', 'Genes', '\t', '\t'),
    ylab.padding = 6,
    xlab.to.xaxis.padding = 0,
    xaxis.lab = list(
        c("",'low',"", "",'high', ""),
        LETTERS[1:25],
        seq(0,5,1),
        NULL
        ),
    yaxis.lab = list(
```

```
        NULL,
        replicate(8, paste(sample(LETTERS, 4, replace = TRUE), collapse = "")),
        NULL,
        seq(0,4,0.05)
        ),
    x.spacing = -0.5,
    y.spacing = c(0,-1),
    xaxis.fontface = 1,
    yaxis.fontface = 1
    );

# Set up plots for complex example

# Dotmap
spot_sizes <- function(x) { 0.5 * abs(x); }
dotmap_dot_colours <- c('red','blue');
spot_colours <- function(x) {
    colours <- rep('white', length(x));
    colours[sign(x) == -1] <- dotmap_dot_colours[1];
    colours[sign(x) ==  1] <- dotmap_dot_colours[2];
    return(colours);
    };

# Dotmap colours
orange <- rgb(249/255, 179/255, 142/255);
blue <- rgb(154/255, 163/255, 242/255);
green <- rgb(177/255, 213/255, 181/255);
bg.colours <- c(green, orange, blue, 'gold', 'skyblue', 'plum');

dotmap <- create.dotmap(
    x = CNA[1:15,1:58],
    bg.data = SNV[1:15,1:58],
    # Set the colour scheme
    colour.scheme = bg.colours,
    # Set the breakpoints for the colour scheme (determined from the data)
    at = c(0,1,2,4,6,7,8),
    # Specify the total number of colours (+1 for the fill colour)
    total.colours = 7,
    col.colour = 'white',
    row.colour = 'white',
    bg.alpha = 1,
    fill.colour = 'grey95',
    spot.size.function = spot_sizes,
    spot.colour.function = spot_colours,
    xaxis.tck = 0,
    xaxis.cex = 0.7,
    yaxis.cex = 0.7,
    xaxis.rot = 90
    );

# Dotmap legend
dotmap_legend <- list(
    legend = list(
```

```
        colours = bg.colours,
        labels = c('Nonsynonymous','Stop Gain','Frameshift deletion',
            'Nonframeshift deletion', 'Splicing', 'Unknown'),
        border = 'white',
        title = 'SNV',
        pch = 15
        ),
    legend = list(
        colours = dotmap_dot_colours,
        labels = c('Gain','Loss'),
        border = 'white',
        title = 'CNA',
        pch = 19
        )
    );

dotmap_legend.grob <- legend.grob(
    legends = dotmap_legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Covariates
cov.colours <- c(
    c('dodgerblue','pink'),
    c('grey','darkseagreen1','seagreen2','springgreen3','springgreen4'),
    c('peachpuff','tan4')
    );

# the heatmap expects numeric data
cov.data <- patient[-c(4:9)];
cov.data[cov.data == 'male'] <- 1;
cov.data[cov.data == 'female'] <- 2;
cov.data[is.na(cov.data)] <- 3;
cov.data[cov.data == 'I'] <- 4;
cov.data[cov.data == 'II'] <- 5;
cov.data[cov.data == 'III'] <- 6;
cov.data[cov.data == 'IV'] <- 7;
cov.data[cov.data == 'MSS'] <- 8;
cov.data[cov.data == 'MSI-High'] <- 9;
cov.data$sex <- as.numeric(cov.data$sex);
cov.data$stage <- as.numeric(cov.data$stage);
cov.data$msi <- as.numeric(cov.data$msi);

covariates <- create.heatmap(
    x = cov.data,
    clustering.method = 'none',
    colour.scheme = as.vector(cov.colours),
    total.colours = 10,
    row.colour = 'white',
    col.colour = 'white',
    grid.row = TRUE,
```

```
        grid.col = TRUE,
        yaxis.tck = 0,
        print.colour.key = FALSE
        );

    # Coviate Legends
    cov_legends <- list(
        legend = list(
            colours = cov.colours[8:9],
            labels = c('MSS','MSI-High'),
            border = 'white',
            title = 'MSI'
            ),
        legend = list(
            colours = cov.colours[3:7],
            labels = c('NA', 'I','II','III','IV'),
            border = 'white',
            title = 'Stage'
            ),
        legend = list(
            colours = cov.colours[1:2],
            labels = c('Male','Female'),
            border = 'white',
            title = 'Sex'
            )
        );

    cov_legend.grob <- legend.grob(
        legends = cov_legends,
        title.just = 'left',
        label.cex = 0.7,
        title.cex = 0.7,
        layout = c(3,1)
        );

    # Multiplot of dotmap and covariates
    create.multiplot(
        # filename = tempfile(pattern = 'Multiplot_Dotmap_Cov', fileext = '.tiff'),
        plot.objects = list(covariates, dotmap),
        main = 'Dotmap & covariates',
        panel.heights = c(1,0.1),
        # Set some of the yat to NULL to let R figure it out
        yat = c(seq(1,15,1), NULL),
        xat = NULL,
        yaxis.lab = list(
            c('Sex','Stage','MSI'),
            rev(rownames(SNV)[1:15])
            ),
        yaxis.cex = 0.7,
        y.spacing = -1,
        legend = list(
            bottom = list(
                x = 0.10,
```

```
            y = 0.50,
            fun = cov_legend.grob
            ),
        right = list(
            x = 0.10,
            y = 0.50,
            fun = dotmap_legend.grob
            )
        ),
    # This parameter must be set for the legend to appear
    print.new.legend = TRUE,
    # Adding spacing for the legend
    bottom.padding = 5
    );

# Add more plots, using more complex layout
# grouped barplot
groupedbar_colours <- c('indianred1','indianred4');

count.SNV <- apply(SNV[1:15,], 2, function(x){length(which(!is.na(x)))});
count.CNA <- apply(CNA[1:15,], 2, function(x){length(which(!(x==0)))});

grouped_data <- data.frame(
    values = c(count.SNV, count.CNA),
    samples = rep(colnames(SNV),2),
    group = rep(c('SNV','CNA'), each = 58)
    );

grouped_barplot <- create.barplot(
    formula = values ~ samples,
    data = grouped_data,
    groups = grouped_data$group,
    col = groupedbar_colours,
    border.col = 'white'
    );

# stacked barplot
col_one <- rgb(255/255, 225/255, 238/255);
col_two <- rgb(244/255, 224/255, 166/255);
col_thr <- rgb(177/255, 211/255, 154/255);
col_fou <- rgb(101/255, 180/255, 162/255);
col_fiv <- rgb(51/255, 106/255, 144/255);
stackedbar_colours <- c(col_one, col_two, col_thr, col_fou, col_fiv, 'orchid4');
stacked_data_labels <- c('C>A/G>T','C>T/G>A','C>G/G>C','T>A/A>T','T>G/A>C', 'T>C/A>G');

stacked_data <- data.frame(
   values = c(patient$prop.CAGT, patient$prop.CTGA, patient$prop.CGGC, patient$prop.TAAT,
        patient$prop.TGAC, patient$prop.TCAG),
    divisions = rep(rownames(patient), 6),
    group = rep(stacked_data_labels, each = 58)
    );

# Generate stacked barplot
```

```
stacked_barplot <- create.barplot(
    formula = values ~ divisions,
    data = stacked_data,
    groups = stacked_data$group,
    stack = TRUE,
    col = stackedbar_colours,
    border.col = 'white'
    );

# barchart legends
stackedbar_legend <- list(
    legend = list(
        colours = rev(stackedbar_colours),
        labels = rev(stacked_data_labels),
        border = 'white'
        )
    );

groupedbar_legend <- list(
    legend = list(
        colours = groupedbar_colours,
        labels = c('CNA','SNV'),
        border = 'white'
        )
    );

groupedbar_legend.grob <- legend.grob(
    legends = groupedbar_legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

stackedbar_legend.grob <- legend.grob(
    legends = stackedbar_legend,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Expression change Segplot
# locate matching genes
rows.to.keep <- which(match(rownames(microarray), rownames(SNV)[1:15], nomatch = 0) > 0);

segplot.data <- data.frame(
    min = apply(microarray[rows.to.keep,1:58], 1, min),
    max = apply(microarray[rows.to.keep,1:58], 1, max),
    median = apply(microarray[rows.to.keep,1:58], 1, median),
    order = seq(1,15,1)
    );

segplot <- create.segplot(
    formula = order ~ min + max,
```

```
        data = segplot.data,
        main = 'Medians',
        centers = segplot.data$median,
        pch = 15
        );

    # Create multiplot
    plots <- list(covariates, dotmap, segplot, stacked_barplot, grouped_barplot);

    create.multiplot(
        # filename = tempfile(pattern = 'Multiplot_Complex', fileext = '.tiff'),
        main = 'Complex',
        # These dimensions make the plot look much more proportional
        width = 12,
        height = 8,
        plot.objects = plots,
        panel.heights = c(0.2, 0.2, 1, 0.1),
        panel.widths = c(1,0.1),
        plot.layout = c(2, 4),
        layout.skip = c(FALSE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE),
        xaxis.lab = list(
            NULL,
            NULL,
            seq(0,14,2),
            NULL,
            NULL),
        yaxis.lab = list(
            c('Sex','Stage','MSI'),
            rownames(SNV)[1:15],
            NULL,
            seq(0.0,1.0,0.2),
            seq(0,16,4)
            ),
        x.spacing = -0.5,
        y.spacing = -1.5,
        xaxis.cex = 0.7,
        yaxis.cex = 0.7,
        xat = list(
            NULL,
            NULL,
            seq(0,10,2.5),
            NULL,
            NULL
            ),
        yat = list(
            seq(1,3,1),
            seq(1,15,1),
            NULL,
            seq(0.0,1.0,0.2),
            seq(0,16,4)
            ),
        ylab.label = c( 'Mutation', 'Proportion','\t','\t','\t','\t','\t'),
        ylab.cex = 0.7,
```

```
    xlab.cex = 0.7,
    xlab.to.xaxis.padding = 2,
    key.bottom.padding = 5,
    bottom.padding = 5,
    right.padding = 8,
    legend = list(
        bottom = list(
            x = 0.10,
            y = 0.50,
            fun = cov_legend.grob
            ),
        inside = list(
            x = 0.91,
            y = 0.96,
            fun = groupedbar_legend.grob
            ),
        inside = list(
            x = 0.91,
            y = 0.86,
            fun = stackedbar_legend.grob
            ),
        left = list(
            fun = dotmap_legend.grob,
            args = list(
                key = list(
                    points = list(
                        pch = c(15,15,19,19)
                        )
                    )
                )
            )
        ),
    print.new.legend = TRUE,
    resolution = 200
    );

# Nature style
create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_Nature_style', fileext = '.tiff'),
    plot.objects = list(simple.heatmap, simple.boxplot),
    main = "Nature style",
    ylab.padding = 7,
    xaxis.cex = 0.7,
    yaxis.cex = 0.7,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = c(expression(paste('italicized ', italic('a'))),
    expression(paste('italicized ', italic('b')))),

    # demonstrating how to create en-dashes
```

```
    xlab.label = c(expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3))),
    resolution = 200
    );

# Create a multiplot with a heatmap, key like legend and barplot

# First create a heatmap object
simple.heatmap <- create.heatmap(patient[, 4:6],
    clustering.method = 'none',
    print.colour.key = FALSE,
    scale=TRUE,
    same.as.matrix = FALSE,
    colour.scheme = c('gray0','grey100'),
    fill.colour = 'grey95'
);


# and a simple bar plot
pvals <- data.frame(
        order = c(1:3),
        pvalue = -log10(c(0.0004, 0.045, 0.0001)),
        stringsAsFactors = FALSE
        )
#create bar plot
simple.bar <- create.barplot(
        formula = order ~ rev(pvalue),
        data = pvals,
        xlimits = c(0,5),
        plot.horizontal=TRUE
        );

# then the covariates heatmap
cov.colours <- c(
    c('dodgerblue','pink'),
    c('grey','darkseagreen1','seagreen2','springgreen3','springgreen4'),
    c('peachpuff','tan4')
    );

# the heatmap expects numeric data
cov.data <- patient[-c(4:9)];
cov.data[cov.data == 'male'] <- 1;
cov.data[cov.data == 'female'] <- 2;
cov.data[is.na(cov.data)] <- 3;
cov.data[cov.data == 'I'] <- 4;
cov.data[cov.data == 'II'] <- 5;
cov.data[cov.data == 'III'] <- 6;
cov.data[cov.data == 'IV'] <- 7;
cov.data[cov.data == 'MSS'] <- 8;
cov.data[cov.data == 'MSI-High'] <- 9;
cov.data$sex <- as.numeric(cov.data$sex);
cov.data$stage <- as.numeric(cov.data$stage);
cov.data$msi <- as.numeric(cov.data$msi);
```

```
covariates <- create.heatmap(
    x = cov.data,
    clustering.method = 'none',
    colour.scheme = as.vector(cov.colours),
    total.colours = 10,
    row.colour = 'white',
    col.colour = 'white',
    grid.row = TRUE,
    grid.col = TRUE,
    yaxis.tck = 0,
    print.colour.key = FALSE
    );
covariates2 <- create.heatmap(
    x = patient[4],
    clustering.method = 'none',
    colour.scheme = c("#00007F", "#007FFF"),
    row.colour = 'white',
    col.colour = 'white',
    grid.row = TRUE,
    grid.col = TRUE,
    yaxis.tck = 0,
    print.colour.key = FALSE
    );

cov_legends <- list(
    legend = list(
        colours = c("white", "black"),
        labels = c('0','2'),
        border = 'grey',
        title = 'Tumour Mass (kg)',
        continuous = TRUE,
height = 3
        ),
    legend = list(
        colours = cov.colours[8:9],
        labels = c('MSS','MSI-High'),
        border = 'white',
        title = 'MSI'
        ),
    legend = list(
        colours = cov.colours[3:7],
        labels = c('NA', 'I','II','III','IV'),
        border = 'white',
        title = 'Stage'
        ),
    legend = list(
        colours = cov.colours[1:2],
        labels = c('Male','Female'),
        border = 'white',
        title = 'Sex'
        ),
    legend = list(
        colours = c("#00007F", "#007FFF"),
```

```
        labels = c('0.09','0.72'),
        border = 'grey',
        title = 'CAGT',
continuous = TRUE,
height = 2,
        width = 3,
angle = -90,
tck = 1,
tck.number = 2,
at = c(0,100)
        )
    );

cov_legend.grob <- legend.grob(
    legends = cov_legends,
    title.just = 'left',
    label.cex = 0.7,
    title.cex = 0.7
    );

# Now bring it was together using multiplot
create.multiplot(
    main = 'multiplot with colour key legend',
    main.cex = 1,
    # filename = tempfile(pattern = 'MultiPlot_With_ColorKey_Legend', fileext = '.tiff'),
    plot.objects = list(covariates, covariates2, simple.heatmap, simple.bar),
    panel.heights = c(1,0.1,0.35),
    panel.widths = c(1,0.25),
    plot.layout = c(2,3),
    layout.skip = c(FALSE, TRUE, FALSE, TRUE,FALSE,FALSE),
    xaxis.alternating = 1,
    # Set some of the yat to NULL to let R figure it out
    yaxis.lab = list(
        c('Sex','Stage','MSI'),
        NULL,
c('one','two','three'),
        NULL
        ),
    xaxis.lab = list(
        NULL,
        NULL,
NULL,
        seq(0,5,1)
    ),
    xat = list(
        NULL,
        NULL,
NULL,
        seq(0,5,1)
    ),
    yaxis.tck = 0,
    xlab.to.xaxis.padding = 0,
    yaxis.cex = 0.5,
```

```
        xaxis.cex = 0.5,
        xlab.cex = 0.75,
        ylab.cex = 0.75,
        xlab.label = c('\t', 'samples', '\t', '    -log10 pval'),
        ylab.label = c("", "Test", "","CAGT",  "covariates"),
        y.spacing = 0,
        x.spacing = 0,
        legend = list(
            left = list(
                x = 0.10,
                y = 0.50,
                fun = cov_legend.grob
                )
            ),
        left.padding = 2.5,
        # This parameter must be set for the legend to appear
        print.new.legend = TRUE
        );

    BarPlotDataRetLabels <- data.frame(x = c("test1","test2","test3","test4"),
       y = c(10000,13000,12000,6700))
    HeatMapDataRetLabels <- matrix(nrow = 4, ncol = 4, data = rnorm(16,1,1))

    bpRet <- create.barplot(
    formula = y~x,
    data = BarPlotDataRetLabels,
    xaxis.lab = NULL,
    xat = 0
    );
    hmRet <- create.heatmap(
    x= HeatMapDataRetLabels,
    yaxis.lab = c("Gene 1", "Gene 2", "Gene 3", "Gene 4"),
    yat = c(1,2,3,4),
    clustering.method = 'none'
    );

    create.multiplot(
    # filename = tempfile(pattern = 'Multiplot_RetrievePlotLabels', fileext = '.tiff'),
    plot.objects = list(hmRet,bpRet,bpRet),
    print.new.legend = TRUE,
    xlab.label = c('Samples'),
    ylab.padding = 12,
    y.spacing = c(0,0),
    panel.heights = c(0.25,1,0.25),
    plot.labels.to.retrieve = c(1,2,3)
    );

    create.multiplot(
       # filename = tempfile(pattern = 'Multiplot_Retrieve_Specefic_Labels', fileext = '.tiff'),
            plot.objects = list(simple.heatmap, simple.boxplot),
            main = "Simple",
            xlab.label = c("Patient Group"),
    xaxis.labels = c("1","Drug Regimen"),
```

```
        # The plotting function throws an error if this is not included
        ylab.label = c("Sugar Level", "Drug Regimen"),
        ylab.padding = 7,
        # Parameters set in the multiplot will override settings in individual plots
        xaxis.cex = 0.7,
        yaxis.cex = 0.7,
yaxis.labels = c(NA,NA),
xat = list(TRUE,TRUE),
yat = list(TRUE,TRUE),
plot.labels.to.retrieve = c(1),
xlimits = list(NULL,c("A","B","C")),
ylimits = list(NULL,c(-3,10))
        );

# Dendrogram provided
dist <- data.frame(
    a = rnorm(100, 1),
    b = rnorm(100, 3),
    c = rnorm(100, 5)
    );

simple.data <- data.frame(
    x = c(dist$a, dist$b, dist$c),
    y = rep(LETTERS[1:3], each = 100)
    );
col.dendrogram <- BoutrosLab.plotting.general::create.dendrogram(
    x = microarray[1:20, 1:20],
    cluster.dimension = 'col'
    );

row.dendrogram <- BoutrosLab.plotting.general::create.dendrogram(
    x = microarray[1:20, 1:20],
    cluster.dimension = 'row'
    );

simple.boxplot <- create.boxplot(
    formula = x ~ y,
    data = simple.data,
    col = 'lightgrey'
    );

simple.heatmap <- create.heatmap(
    x = microarray[1:20, 1:20],
    main = 'Dendrogram provided',
    xlab.label = 'Genes',
    ylab.label = 'Samples',
    xaxis.lab = NA,
    yaxis.lab = 1:20,
    xaxis.cex = 0.75,
    yaxis.cex = 0.75,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    colourkey.cex = 1,
```

```
        colourkey.labels.at = seq(2,12,1),
        colour.alpha = 'automatic',
      # note: row/column dendrograms are switched because the function inverts rows and columns
        clustering.method = 'none',
        row.dendrogram = col.dendrogram,
        col.dendrogram = row.dendrogram,
        # Adjusting the size of the dendrogram
        right.dendrogram.size = 3,
        top.dendrogram.size = 2.5,
        description = 'Heatmap created using BoutrosLab.plotting.general'
        );

    legend <- legend.grob(
        list(
            legend = list(
                colours = default.colours(4),
                title = "Batch",
                labels = LETTERS[1:4],
                size = 3,
                title.cex = 1,
                label.cex = 1,
                border = 'black'
                ),
            legend = list(
                colours = c("lightblue","dodgerblue2","dodgerblue4"),
                title = "Grade",
                labels = c("Low","Normal","High"),
                size = 3,
                title.cex = 1,
                label.cex = 1,
                border = 'black'
                ),
            legend = list(
                colours = c("grey","coral1"),
                title = "Biomarker",
                labels = c("Not present","Present"),
                size = 3,
                title.cex = 1,
                label.cex = 1,
                border = 'black'
                )
            ),
        title.just = 'left'
        );
    create.multiplot(
            # filename = tempfile(pattern = 'MultiPlot_getDendrograms', fileext = '.tiff'),
            plot.objects = list(simple.heatmap, simple.boxplot),
            main = "Simple",
            xlab.label = c("Patient Group"),
    y.spacing = 3,
            # The plotting function throws an error if this is not included
            ylab.label = c("Sugar Level", "Drug Regimen"),
            ylab.padding = 7,
```

```
        # Parameters set in the multiplot will override settings in individual plots
        xaxis.cex = 0.7,
        yaxis.cex = 0.7,
yaxis.lab = list(
c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20),
c(-2,-1,0,1,2,3,4,5)
),
xaxis.lab = list(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, 16,17,18,19,20),c(1,2,3)),
xaxis.rot = 45,
xaxis.rot.top = 90,
     legend = list(right = list(fun = legend)),
print.new.legend = TRUE,
get.dendrogram.from = 1,
dendrogram.right.size = 0.40, dendrogram.right.x = 29, dendrogram.right.y = 67,
dendrogram.top.size = 1, dendrogram.top.x = 110, dendrogram.top.y = -180
        );
```

`create.polygonplot` *Make a polygonplot*

Description

Takes a data.frame and creates a polygon

Usage

```
create.polygonplot(
formula,
data,
filename = NULL,
groups = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
max,
min,
col = 'white',
alpha = 0.5,
border.col = 'black',
strip.col = 'white',
strip.cex = 1,
type = 'p',
cex = 0.75,
pch = 19,
lwd = 1,
```

```
    lty = 1,
    axes.lwd = 1,
    xlab.label = tail(sub('~', '', formula[-2]), 1),
    ylab.label = tail(sub('~', '', formula[-3]), 1),
    xlab.cex = 2,
    ylab.cex = 2,
    xlab.col = 'black',
    ylab.col = 'black',
    xlab.top.label = NULL,
    xlab.top.cex = 2,
    xlab.top.col = 'black',
    xlab.top.just = 'center',
    xlab.top.x = 0.5,
    xlab.top.y = 0,
    xaxis.lab = TRUE,
    yaxis.lab = TRUE,
    xaxis.cex = 1.5,
    yaxis.cex = 1.5,
    xaxis.rot = 0,
    yaxis.rot = 0,
    xaxis.log = FALSE,
    yaxis.log = FALSE,
    xaxis.fontface = 'bold',
    yaxis.fontface = 'bold',
    xaxis.col = 'black',
    yaxis.col = 'black',
    xaxis.tck = 1,
    yaxis.tck = 1,
    xlimits = NULL,
    ylimits = NULL,
    xat = TRUE,
    yat = TRUE,
    layout = NULL,
    as.table = FALSE,
    x.spacing = 0,
    y.spacing = 0,
    x.relation = 'same',
    y.relation = 'same',
    top.padding = 0.5,
    bottom.padding = 2,
    right.padding = 1,
    left.padding = 2,
    ylab.axis.padding = 0,
    add.border = FALSE,
    add.xy.border = NULL,
    add.median = FALSE,
    median.lty = 3,
    median.lwd = 1.5,
```

```
    use.loess.border = FALSE,
    use.loess.median = FALSE,
    median = NULL,
    median.col = 'black',
    extra.points = NULL,
    extra.points.pch = 21,
    extra.points.type = 'p',
    extra.points.col = 'black',
    extra.points.fill = 'white',
    extra.points.cex = 1,
    add.rectangle = FALSE,
    xleft.rectangle = NULL,
    ybottom.rectangle = NULL,
    xright.rectangle = NULL,
    ytop.rectangle = NULL,
    col.rectangle = 'transparent',
    alpha.rectangle = 1,
    xgrid.at = xat,
    ygrid.at = yat,
    grid.lty = 1,
    grid.col = 'grey',
    grid.lwd = 0.3,
    add.xyline = FALSE,
    xyline.col = 'black',
    xyline.lwd = 1,
    xyline.lty = 1,
    abline.h = NULL,
    abline.v = NULL,
    abline.col = 'black',
    abline.lwd = 1,
    abline.lty = 1,
    add.text = FALSE,
    text.labels = NULL,
    text.x = NULL,
    text.y = NULL,
    text.col = 'black',
    text.cex = 1,
    text.fontface = 'bold',
    key = NULL,
    legend = NULL,
    height = 6,
    width = 6,
    size.units = 'in',
    resolution = 1600,
    enable.warnings = FALSE,
    description = 'Created with BoutrosLab.plotting.general',
    style = 'BoutrosLab',
    preload.default = 'custom',
```

```
use.legacy.settings = FALSE,
inside.legend.auto = FALSE
);
```

Arguments

`formula`	The formula used to extract the boxplot components from the data-frame
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`groups`	The grouping variable in the data-frame
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title
`max`	Max values for polygon
`min`	Min values for polygon
`col`	Fill colour of polygon, defaults to white
`alpha`	Transparency of polygons when several are plotted, defaults to 0.5.
`border.col`	Border colour(s) of polygon(s), defaults to black
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`type`	Plot type
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`lwd`	Specifies line width, defaults to 1
`lty`	Specifies line style, defaults to 1 (solid)
`axes.lwd`	Thickness of width of axes lines
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label

`xlab.top.y`	The y location of the top y-axis label
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2
`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xlimits`	Two-element vector giving the x-axis limits
`ylimits`	Two-element vector giving the y-axis limits
`xat`	Vector listing where the x-axis labels should be drawn
`yat`	Vector listing where the y-axis labels should be drawn
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details

.

`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`top.padding`	A number specifying the distance to the top margin, defaults to 0.5
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 2
`right.padding`	A number specifying the distance to the right margin, defaults to 1
`left.padding`	A number specifying the distance to the left margin, defaults to 2
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 0

,

`add.border`	Add xy border to polygon, default is FALSE
`add.xy.border`	DEPRECATED: Use ‘add.border‘ argument instead
`add.median`	Add median line, default is FALSE
`median.lty`	Median line type
`median.lwd`	Median line width, defaults to 1.5
`use.loess.border`	Use loess curve for border instead of max/min values, default is FALSE
`use.loess.median`	Use loess curve for median values, default is FALSE
`median`	Median values for median line
`median.col`	Median line colour, default is black
`extra.points`	If not set to NULL (default), add a set of extra points to the plot. A list of two numeric vectors named “x” and “y” giving the co-ordinates of the points to be added
`extra.points.pch`	A vector specifying the types of extra points to add to the plot. Defaults to 21
`extra.points.type`	A vector specifying the plot type. Defaults to “p”
`extra.points.col`	A vector specifying the border colours of the extra points to add to the plot. Defaults to “black”
`extra.points.fill`	A vector specifying the fill colours of the extra points to add to the plot. Defaults to “white”
`extra.points.cex`	A vector specifying the sizes of the extra points to add to the plot. Defaults to 1
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle
`xgrid.at`	A vector listing the co-ordinates at which vertical grid-lines should be drawn. Default suppresses drawing of vertical grid-lines
`ygrid.at`	A vector listing the co-ordinates at which horizontal grid-lines should be drawn. Default suppresses drawing of horizontal grid-lines
`grid.lty`	Specifies the line type to use for the grid-lines. Defaults to 1 (solid lines)

`grid.col`	Specifies the colour to use for the grid-lines. Defaults to “grey”
`grid.lwd`	Specifies the width of the grid-lines. Defaults to 0.3
`add.xyline`	Allow y=x line to be drawn, default is FALSE
`xyline.col`	y=x line colour, defaults to black
`xyline.lwd`	Specifies y=x line width, defaults to 1
`xyline.lty`	Specifies y=x line style, defaults to 1 (solid)
`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.v`	Allow vertical line to be drawn, default to NULL
`abline.col`	Horizontal line colour, defaults to black
`abline.lwd`	Specifies horizontal line width, defaults to 1
`abline.lty`	Specifies horizontal line style, defaults to 1 (solid)
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed
`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`key`	Add a key to the plot. See xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Denise Mak

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

temp  <- matrix(runif(1010), ncol = 10) + sort(runif(101));

simple.data <- data.frame(
    x = 0:100,
    max = apply(temp, 1, max),
    min = apply(temp, 1, min)
    );

create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Simple', fileext = '.tiff'),
    formula = NA ~ x,
    data = simple.data,
    max = simple.data$max,
    min = simple.data$min,
    main = 'Simple',
    xlimits = c(0,100),
    ylimits = c (0,2),
    col = default.colours(1),
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Compare two genes across increasing numbers of samples
data1 <- microarray[1,1:58];
data2 <- microarray[2,1:58];

gene1 <- as.data.frame(matrix(nrow = 58, ncol = 58));
```

```
gene2 <- as.data.frame(matrix(nrow = 58, ncol = 58));

fill.matrix <- function(x, gene, data){
    for(i in x){
        gene[i, 1:i] <- rep(NA, i);
        gene[i, i:58] <- rep(as.numeric(data[i]), 58-i+1);
        }
    return(gene);
    };

gene1 <- fill.matrix(1:58, gene1, data1);
gene1 <- t(matrix(unlist(gene1), ncol = 58, byrow = TRUE));

gene2 <- fill.matrix(1:58, gene2, data2);
gene2 <- t(matrix(unlist(gene2), ncol = 58, byrow = TRUE));

# Set up the data
polygon.data.gene1 <- data.frame(
    x = 1:58,
    max = apply(gene1, 2, function(x) {max(x, na.rm = TRUE)}),
    median = apply(gene1, 2, function(x) {median(x, na.rm = TRUE)}),
    min = apply(gene1, 2, function(x) {min(x, na.rm = TRUE)}),
    set = rownames(microarray[1,]),
    extra = apply(microarray[1:58], 2, function(x) {median(x)})
    );

polygon.data.two.genes <- rbind(
    polygon.data.gene1,
    data.frame(
        x = 1:58,
        max = apply(gene2, 2, function(x) {max(x, na.rm = TRUE)}),
        median = apply(gene2, 2, function(x) {median(x, na.rm = TRUE)}),
        min = apply(gene2, 2, function(x) {min(x, na.rm = TRUE)}),
        set = rownames(microarray[2,]),
        extra = apply(microarray[1:58], 2, function(x) {median(x)})
        )
    )

# Minimal Input
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Minimal_Input', fileext = '.tiff'),
    formula = NA ~ x,
    data = polygon.data.gene1,
    max = polygon.data.gene1$max,
    min = polygon.data.gene1$min,
    main = 'Minimal input',
    xlimits = c(0,58),
    ylimits = c (2,5),
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes & Labels
```

```
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Axes_Labels', fileext = '.tiff'),
    formula = NA ~ x,
    data = polygon.data.gene1,
    max = polygon.data.gene1$max,
    min = polygon.data.gene1$min,
    main = 'Axes & labels',
    xlimits = c(0,58),
    ylimits = c (0,10),
    # Axes & Labels
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 10, 2),
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colour
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Colour', fileext = '.tiff'),
    formula = NA ~ x,
    data = polygon.data.gene1,
    max = polygon.data.gene1$max,
    min = polygon.data.gene1$min,
    main = 'Colour',
    xlimits = c(0,58),
    ylimits = c (0,10),
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 10, 2),
    # Colour
    col = default.colours(1),
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Add median line and points
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Median_Points', fileext = '.tiff'),
```

```
    formula = NA ~ x,
    data = polygon.data.gene1,
    max = polygon.data.gene1$max,
    min = polygon.data.gene1$min,
    # Median
    median = polygon.data.gene1$median,
    add.median = TRUE,
    main = 'Plotting character',
    xlimits = c(0,58),
    ylimits = c (0,10),
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 10, 2),
    col = default.colours(1),
    # border points
    add.border = TRUE,
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Additional Data
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Extra_Data', fileext = '.tiff'),
    formula = NA ~ x,
    # divide data
    groups = set,
    data = polygon.data.two.genes,
    max = polygon.data.two.genes$max,
    min = polygon.data.two.genes$min,
    main = 'Two data sets',
    median = polygon.data.two.genes$median,
    add.median = TRUE,
    xlimits = c(0,58),
    ylimits = c (0,15),
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 14, 2),
    # Increasing number of colours
```

```
        col = default.colours(2),
        description = 'Polygon created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Legend
    create.polygonplot(
        # filename = tempfile(pattern = 'Polygon_Legend', fileext = '.tiff'),
        formula = NA ~ x,
        groups = set,
        data = polygon.data.two.genes,
        max = polygon.data.two.genes$max,
        min = polygon.data.two.genes$min,
        main = 'Legend',
        median = polygon.data.two.genes$median,
        add.median = TRUE,
        xlimits = c(0,58),
        ylimits = c (0,15),
        xlab.label = 'Samples',
        ylab.label = 'Value',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xat = seq(0, 58, 5),
        yat = seq(0, 14, 2),
        col = default.colours(2),
        # Adding legend
        key = list(
            text = list(
                lab = rownames(microarray[1:2,]),
                cex = 0.8,
                col = 'black'
                ),
            points = list(
                pch = 15,
                col = default.colours(2),
                cex = 2
                ),
            x = 0.04,
            y = 0.93,
            padding.text = 3,
            columns = 1
            ),
        description = 'Polygon created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Panel Organiation
    create.polygonplot(
        # filename = tempfile(pattern = 'Polygon_Panel', fileext = '.tiff'),
```

```
    # divide data
    formula = NA ~ x | set,
    data = polygon.data.two.genes,
    max = polygon.data.two.genes$max,
    min = polygon.data.two.genes$min,
    main = 'Panel',
    median = polygon.data.two.genes$median,
    add.median = TRUE,
    xlimits = c(0,58),
    ylimits = c (0,15),
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 14, 2),
    col = default.colours(1),
    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Extra Points
create.polygonplot(
    # filename = tempfile(pattern = 'Polygon_Extra_Points', fileext = '.tiff'),
    formula = NA ~ x,
    groups = set,
    data = polygon.data.two.genes,
    max = polygon.data.two.genes$max,
    min = polygon.data.two.genes$min,
    main = 'Extra points',
    median = polygon.data.two.genes$median,
    add.median = TRUE,
    xlimits = c(0,58),
    ylimits = c (0,15),
    xlab.label = 'Samples',
    ylab.label = 'Value',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xat = seq(0, 58, 5),
    yat = seq(0, 14, 2),
    col = default.colours(2),
    # Add to legend
    key = list(
        text = list(
            lab = c(rownames(microarray[1:2,]), 'All genes'),
```

```
                cex = 0.8,
                col = 'black'
                ),
            points = list(
                pch = c(15, 15, 3),
                col = c(default.colours(2), 'red'),
                cex = c(2, 2, 0.7)
                ),
            x = 0.04,
            y = 0.93,
            padding.text = 3,
            columns = 1
            ),
        # Extra points
        extra.points = list(
            x = polygon.data.two.genes$x,
            y = polygon.data.two.genes$extra
            ),
        extra.points.col = 'red',
        extra.points.pch = 3,
        extra.points.type = c('p', 'l'),
        extra.points.cex = 0.7,
        description = 'Polygon created by BoutrosLab.plotting.general',
        resolution = 200
        );

    # Nature style
    create.polygonplot(
        # filename = tempfile(pattern = 'Polygon_Nature_style', fileext = '.tiff'),
        formula = NA ~ x,
        groups = set,
        data = polygon.data.two.genes,
        max = polygon.data.two.genes$max,
        min = polygon.data.two.genes$min,
        main = 'Nature style',
        median = polygon.data.two.genes$median,
        add.median = TRUE,
        xlimits = c(0,58),
        ylimits = c (0,15),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xat = seq(0, 58, 5),
        yat = seq(0, 14, 2),
        col = default.colours(2),
        # Adding legend
        key = list(
            text = list(
                lab = rownames(microarray[1:2,]),
                cex = 0.8,
```

```
            col = 'black'
            ),
        points = list(
            pch = 15,
            col = default.colours(2),
            cex = 2
            ),
        x = 0.04,
        y = 0.93,
        padding.text = 3,
        columns = 1
        ),

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Polygon created by BoutrosLab.plotting.general',
    resolution = 1200
    );
```

`create.qqplot.comparison` *Make a quantile-quantile plot of two samples*

Description

Takes two samples and creates a qq plot for comparing two distributions, possibly conditioned on other variables

Usage

```
create.qqplot.comparison(
x,
data = NULL,
filename = NULL,
groups = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
aspect = 'fill',
```

```
    prepanel = NULL,
    xlab.label = NULL,
    ylab.label = NULL,
    xlab.cex = 2,
    ylab.cex = 2,
    xlab.col = 'black',
    ylab.col = 'black',
    xlimits = NULL,
    ylimits = NULL,
    xat = TRUE,
    yat = TRUE,
    xaxis.lab = NA,
    yaxis.lab = NA,
    xaxis.cex = 1.5,
    yaxis.cex = 1.5,
    xaxis.fontface = 'bold',
    yaxis.fontface = 'bold',
    xaxis.log = FALSE,
    yaxis.log = FALSE,
    xaxis.rot = 0,
    yaxis.rot = 0,
    xaxis.col = 'black',
    yaxis.col = 'black',
    xaxis.tck = 1,
    yaxis.tck = 1,
    xlab.top.label = NULL,
    xlab.top.cex = 2,
    xlab.top.col = 'black',
    xlab.top.just = 'center',
    xlab.top.x = 0.5,
    xlab.top.y = 0,
    add.grid = FALSE,
    xgrid.at = xat,
    ygrid.at = yat,
    type = 'p',
    cex = 0.75,
    pch = 19,
    col = 'black',
    lwd = 1,
    lty = 1,
    axes.lwd = 2.25,
    key = list(text = list(lab = c(''))),
    legend = NULL,
    add.rectangle = FALSE,
    xleft.rectangle = NULL,
    ybottom.rectangle = NULL,
    xright.rectangle = NULL,
    ytop.rectangle = NULL,
```

```
col.rectangle = 'transparent',
alpha.rectangle = 1,
top.padding = 3,
bottom.padding = 0.7,
left.padding = 0.5,
right.padding = 0.1,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
inside.legend.auto = FALSE
);
```

Arguments

`x`	A formula or a list of two numeric vectors
`data`	An optional data source if x is a formula
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`aspect`	This argument controls the physical aspect ratio of the panels, defaults to “fill”
`prepanel`	A function that takes the same arguments as the “panel”
`add.grid`	Default manner of drawing grid lines - for custom grids, use type = c(’p’,’g’) and set the xat, yat, xgrid.at, ygrid.at parameters
`groups`	The grouping variable in the data-frame
`main`	The main plot title
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`main.cex`	Size of the overall plot title, defaults to 3
`xlab.cex`	Size of x-axis label, defaults to 2.5
`ylab.cex`	Size of y-axis label, defaults to 2.5
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label

`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`xgrid.at`	Vector listing where the x-axis grid lines should be drawn, defaults to xat
`ygrid.at`	Vector listing where the y-axis grid lines should be drawn, defaults to yat
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.cex`	Size of x-axis scales, defaults to 1.5
`yaxis.cex`	Size of y-axis scales, defaults to 1.5
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`type`	Plot type
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`col`	Point/line colour
`lwd`	Specifies line width, defaults to 1
`lty`	Specifies line style, defaults to 1 (solid)
`axes.lwd`	Thickness of width of axes lines
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`top.padding`	A number giving the top padding in multiples of the lattice default
`bottom.padding`	A number giving the bottom padding in multiples of the lattice default
`left.padding`	A number giving the left padding in multiples of the lattice default

`right.padding`	A number giving the right padding in multiples of the lattice default
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`style`	defaults to "BoutrosLab", also accepts "Nature", which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Author(s)

Ying Wu

See Also

`qq`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Simple', fileext = '.tiff'),
    x = list(rnorm(100), rnorm(100)),
    resolution = 50
    );

# Minimal Input
create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Minimal_Input', fileext = '.tiff'),
    x = list(microarray[1:500,2], microarray[1:500,2]),
    main = 'Minimal input',
    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Axes & Labels
create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Axes_Labels', fileext = '.tiff'),
    x = list(microarray[1:500,2], microarray[1:500,2]),
    main = 'Axes & labels',
    # adding axes and labels
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # adding grid for good measure
    add.grid = TRUE,
    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Formula input

# 'Formula' format of data
chr.locations <- microarray$Chr[1:500];
chr.locations <- replace(chr.locations, which(chr.locations == 1), 'Chromosome 1');
chr.locations <- replace(chr.locations, which(chr.locations == 2), 'Chromosome 2');

qqplot.data <- data.frame(
    sample = c(rep('Sample 1', 500), rep('Sample 2', 500)),
    value = c(microarray[1:500,1], microarray[1:500,2]),
    chr = chr.locations
    );


create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Formula', fileext = '.tiff'),
    # Using a different input method
    x = sample ~ value,
```

```
    data = qqplot.data,
    main = 'Formula input',
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xaxis.lab = seq(0, 15, 5),
    yaxis.lab = seq(0, 15, 5),
    xlimits = c(0, 17),
    ylimits = c(0, 17),
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    add.grid = TRUE,
    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 200
    );


# Groups & Legend
create.qqplot.comparison(
 # filename = tempfile(pattern = 'QQcomparison_Groups_Legend', fileext = '.tiff'),
    x = sample ~ value,
    data = qqplot.data,
    # Using fake grouping for the sake of illustration
    groups = qqplot.data$chr,
    # Set colours to differente the gruops
    col = default.colours(3),
    # Setting different plotting characters
    pch = c(15, 16),
    main = 'Groups & legend',
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    add.grid = TRUE,
    # Adding legend to explain groups
    key = list(
        text = list(
            lab = c('1','2'),
            cex = 1.5,
            col = 'black'
            ),
        points = list(
            pch = c(15, 16),
            col = default.colours(2),
            cex = 1
            ),
        x = 0.04,
        y = 0.95,
        padding.text = 2
        ),
    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# Multiple qq plots
create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Multiple', fileext = '.tiff'),
    x = sample ~ value | chr,
    data = qqplot.data,
    main = 'Multiple plots',
    xlab.label = 'Sample 1',
    ylab.label = 'Sample 2',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    add.grid = TRUE,
    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.qqplot.comparison(
    # filename = tempfile(pattern = 'QQcomparison_Nature_style', fileext = '.tiff'),
    x = sample ~ value,
    data = qqplot.data,
    main = 'Nature style',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    add.grid = TRUE,

   # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'QQplot comparison created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

`create.qqplot.fit` *Make a quantile-quantile plot of a sample*

Description

Takes a sample and creates a qq plot against a theoretical distribution, possibly conditioned on other variables.

Usage

```
create.qqplot.fit(
```

```
x,
data = NA,
filename = NULL,
groups = NULL,
confidence.bands = FALSE,
conf = 0.95,
        confidence.method = 'both',
reference.line.method = 'quartiles',
distribution = qnorm,
aspect = 'fill',
        prepanel = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
        xlab.label = NULL,
ylab.label = NULL,
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
        xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
        xlab.top.y = 0,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.lab = NA,
yaxis.lab = NA,
        xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.fontface = 'bold',
        yaxis.fontface = 'bold',
xaxis.log = FALSE,
yaxis.log = FALSE,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.tck = 1,
        yaxis.tck = 1,
add.grid = FALSE,
xgrid.at = xat,
```

```
ygrid.at = yat,
type = 'p',
cex = 0.75,
pch = 19,
col = 'black',
        col.line = 'grey',
lwd = 2,
lty = 1,
axes.lwd = 2.25,
key = list(text = list(lab = c(''))),
legend = NULL,
        add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
        ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
top.padding = 3,
bottom.padding = 0.7,
        left.padding = 0.5,
right.padding = 0.1,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
        enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
        style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
inside.legend.auto = FALSE
);
```

Arguments

`x`	A formula or a numeric vector
`data`	An optional data source if x is a formula
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`groups`	The grouping variable in the data-frame
`confidence.bands`	Add confidence bands or not, default to FALSE. Note that in this function, the confidence band can only be added to a single plot, not for multi-qq plot.
`conf`	Confidence level, default to 0.95
`confidence.method`	Methods used to draw confidence bands: “simultaneous”, “pointwise”, “both”, defaults to “both”.

`reference.line.method`	Methods used to draw reference line and must be one of "quartiles"(default), "diagonal", "robust". "quartiles" will draw a line across 1/4 and 3/4 quantiles, "diagonal" will draw a 0-1 line, "robust" will draw a best fit line basing on linear model. Note: for multi-panel plot, only the default one is applicable.
`distribution`	A quantile function that takes a vector of probabilities as argument and produces the corresponding quantiles from a theoretical distribution, defaults to "qnorm", that is normal distribution.
`aspect`	This argument controls the physical aspect ratio of the panels, defaults to "fill"
`prepanel`	A function that takes the same arguments as the "panel"
`main`	The main plot title
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of the overall plot title, defaults to 3
`xlab.label`	x-axis title
`ylab.label`	y-axis title
`xlab.cex`	Size of x-axis label, defaults to 2.5
`ylab.cex`	Size of y-axis label, defaults to 2.5
`xlab.col`	Colour of the x-axis label, defaults to "black"
`ylab.col`	Colour of the y-axis label, defaults to "black"
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis scales, defaults to 1.5
`yaxis.cex`	Size of y-axis scales, defaults to 1.5
`xaxis.col`	Colour of the x-axis tick labels, defaults to "black"
`yaxis.col`	Colour of the y-axis tick labels, defaults to "black"
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales

`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 1
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`add.grid`	Default manner of drawing grid lines
`xgrid.at`	Vector listing where the x-axis grid lines should be drawn, defaults to xat
`ygrid.at`	Vector listing where the y-axis grid lines should be drawn, defaults to yat
`type`	Plot type
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`col`	Point colour
`col.line`	QQ line colour, defaults to grey
`lwd`	Specifies line width, defaults to 2
`lty`	Specifies line style, defaults to 1 (solid)
`axes.lwd`	Thickness of width of axes lines
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`top.padding`	A number giving the top padding in multiples of the lattice default
`bottom.padding`	A number giving the bottom padding in multiples of the lattice default
`left.padding`	A number giving the left padding in multiples of the lattice default
`right.padding`	A number giving the right padding in multiples of the lattice default
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches

`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL.
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

Note that the confidence band only works for a single panel qq plot, not for grouped data and multi-qq plot. Why? What’s missing?

Author(s)

Ying Wu

See Also

`qqmath`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Simple', fileext = '.tiff'),
    x = rnorm(300),
    # choosing to compare against a uniform distribution
    distribution = qunif,
    resolution = 100
    );

# Minimal Input
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Minimal_Input', fileext = '.tiff'),
    x = microarray[1:500,1],
    # choosing to compare against a uniform distribution
    distribution = qunif,
```

```
    main = 'Minimal input',
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes and Labels
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Axes_Labels', fileext = '.tiff'),
    x = microarray[1:500,1],
    distribution = qunif,
    main = 'Axes & labels',
    # Adding axes labels
    xlab.label = 'qunif',
    ylab.label = 'sample values',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xaxis.cex = 1,
    yaxis.cex = 1,
    add.grid = TRUE,
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Confidence bands
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Confidence_Bands', fileext = '.tiff'),
    x = microarray[1:500,1],
    distribution = qunif,
    main = 'Confidence bands',
    xlab.label = 'qunif',
    ylab.label = 'sample values',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xaxis.cex = 1,
    yaxis.cex = 1,
    add.grid = TRUE,
    # Adding confidence bands (auto-generates legend)
    confidence.bands = TRUE,
    confidence.method = 'both',
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Multiple qq plot conditioned on a variable
# 'Formula' format of data
chr.locations <- microarray$Chr[1:500];
chr.locations <- replace(chr.locations, which(chr.locations == 1), 'Chromosome 1');
chr.locations <- replace(chr.locations, which(chr.locations == 2), 'Chromosome 2');
```

```
qqplot.data <- data.frame(
    value = microarray[1:500,1],
    chr = chr.locations
    );

create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Multiple', fileext = '.tiff'),
    x = ~ value | chr,
    data = qqplot.data,
    distribution = qunif,
    main = 'Multiple plots',
    xlab.label = 'qunif',
    ylab.label = 'sample values',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xaxis.cex = 1,
    yaxis.cex = 1,
    add.grid = TRUE,
    confidence.bands = TRUE,
    confidence.method = 'simultaneous',
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Grouped qq plot
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Grouped', fileext = '.tiff'),
    x = ~ value,
    data = qqplot.data,
    # Adding groups
    groups = qqplot.data$chr,
    # Colouring groups
    col = default.colours(2),
    # Setting different plotting characters
    pch = c(15, 19),
    distribution = qunif,
    main = 'Grouped & legend',
    xlab.label = 'qunif',
    ylab.label = 'sample values',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xaxis.cex = 1,
    yaxis.cex = 1,
    add.grid = TRUE,
    confidence.bands = TRUE,
    confidence.method = 'simultaneous',
    # Adding legend for groups
    key = list(
```

```
        text = list(
            lab = c('1','2'),
            cex = 1,
            col = 'black'
            ),
        points = list(
            pch = c(15, 19),
            col = default.colours(2),
            cex = 1
            ),
        x = 0.04,
        y = 0.95,
        padding.text = 2
        ),
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Correlation Key
create.qqplot.fit(
    # filename = tempfile(pattern = 'QQfit_Correlation_Key', fileext = '.tiff'),
    x = ~ value,
    data = qqplot.data,
    groups = qqplot.data$chr,
    col = default.colours(2),
    pch = c(15, 19),
    distribution = qunif,
    main = 'Correlation key',
    xlab.label = 'qunif',
    ylab.label = 'sample values',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xaxis.cex = 1,
    yaxis.cex = 1,
    add.grid = TRUE,
    confidence.bands = TRUE,
    confidence.method = 'simultaneous',
    # Adjusting legend to contain multiple keys
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                        text = list(
                            lab = c('1','2'),
                            cex = 1,
                            col = 'black'
                            ),
                        points = list(
                            pch = c(15, 19),
                            col = default.colours(2),
```

```
                                cex = 1
                                ),
                            x = 0.14,
                            y = 0.80,
                            padding.text = 2
                        )
                    )
                ),
            inside = list(
                fun = draw.key,
                args = list(
                    key = get.corr.key(
                        x = runif(500),
                        y = qqplot.data$value,
                        label.items = c('spearman', 'kendall','beta1'),
                        alpha.background = 0,
                        key.cex = 1
                        )
                    ),
                x = 0.75,
                y = 0.20,
                corner = c(0,1)
                )
            ),
        description = 'QQplot fit created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Nature style
    create.qqplot.fit(
        # filename = tempfile(pattern = 'QQfit_Nature_style', fileext = '.tiff'),
        x = microarray[1:500,1],
        distribution = qunif,
        main = 'Nature style',
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xaxis.cex = 1,
        yaxis.cex = 1,
        add.grid = TRUE,
        confidence.bands = TRUE,
        confidence.method = 'both',

        # set style to Nature
        style = 'Nature',

        # demonstrating how to italicize character variables
        ylab.label = expression(paste('italicized ', italic('a'))),

        # demonstrating how to create en-dashes
        xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),
```

```
    description = 'QQplot fit created by BoutrosLab.plotting.general',
    resolution = 1200
    );
```

`create.qqplot.fit.confidence.interval` *Create the confidence bands for a one-sample qq plot*

Description

Returns the values of constructing the confidence bands for a one-sample qq plot

Usage

```
create.qqplot.fit.confidence.interval(x, distribution = qnorm, conf = 0.95,
    conf.method = "both", reference.line.method = "quartiles");
```

Arguments

`x`	A numeric vector
`distribution`	A quantile function that takes a vector of probabilities as argument and produces the corresponding quantiles from a theoretical distribution, defaults to "qnorm", that is normal distribution.
`conf`	Confidence level, default to 0.95
`conf.method`	Methods used to draw confidence bands and must be one of "simultaneous", "pointwise", "both"(default).
`reference.line.method`	Methods used to draw reference line and must be one of "quartiles"(default), "diagonal", "robust".

Value

Returns the values of creating the upper and lower bands for the qq plot.

Warning

Note that this function works only for a single panel qq plot, not for grouped data and multi-qq plot.

Author(s)

Ying Wu

Examples

```
tmp.x <- rnorm(100);

tmp.confidence.interval <- create.qqplot.fit.confidence.interval(tmp.x);

qqnorm(tmp.x);
qqline(tmp.x);
lines(tmp.confidence.interval$z, tmp.confidence.interval$upper.pw, lty = 2, col = "brown");
lines(tmp.confidence.interval$z, tmp.confidence.interval$lower.pw, lty = 2, col = "brown");
lines(tmp.confidence.interval$z[tmp.confidence.interval$u],
    tmp.confidence.interval$upper.sim, lty = 2, col = "blue");
lines(tmp.confidence.interval$z[tmp.confidence.interval$l],
    tmp.confidence.interval$lower.sim, lty = 2, col = "blue");

legend(1, -1.5, c("simultaneous", "pointwise"), col = c("blue", "brown"), lty = 2, bty = "n");
```

`create.scatterplot` *Make a scatterplot*

Description

Takes a data.frame and creates a scatterplot

Usage

```
create.scatterplot(
formula,
data,
filename = NULL,
groups = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
        main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
        xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
        xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
        xlab.top.y = 0,
```

```
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
xaxis.lab = NA,
yaxis.lab = NA,
xaxis.log = FALSE,
yaxis.log = FALSE,
        xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.tck = c(1,1),
yaxis.tck = c(1,1),
add.grid = FALSE,
xgrid.at = xat,
        ygrid.at = yat,
grid.colour = NULL,
horizontal = FALSE,
type = 'p',
cex = 0.75,
pch = 19,
col = 'black',
col.border = 'black',
lwd = 1,
lty = 1,
alpha = 1,
axes.lwd = 1,
strip.col = 'white',
strip.cex = 1,
strip.fontface = 'bold',
y.error.up = NULL,
y.error.down = y.error.up,
x.error.right = NULL,
x.error.left = x.error.right,
y.error.bar.col = 'black',
x.error.bar.col = y.error.bar.col,
error.whisker.angle = 90,
error.bar.lwd = 1,
error.bar.length = 0.1,
key = list(text = list(lab = c(''))),
legend = NULL,
top.padding = 0.1,
bottom.padding = 0.7,
```

```
right.padding = 0.1,
left.padding = 0.5,
key.top = 0.1,
key.left.padding = 0,
ylab.axis.padding = 1,
axis.key.padding = 1,
layout = NULL,
as.table = FALSE,
x.spacing = 0,
y.spacing = 0,
x.relation = 'same',
y.relation = 'same',
add.axes = FALSE,
axes.lty = 'dashed',
add.xyline = FALSE,
xyline.col = 'black',
xyline.lwd = 1,
xyline.lty = 1,
abline.h = NULL,
abline.v = NULL,
abline.col = 'black',
abline.lwd = 1,
abline.lty = 1,
add.curves = FALSE,
curves.exprs = NULL,
curves.from = min(data, na.rm = TRUE),
curves.to = max(data, na.rm = TRUE),
curves.col = 'black',
curves.lwd = 2,
curves.lty = 1,
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
add.points = FALSE,
points.x = NULL,
points.y = NULL,
points.pch = 19,
points.col = 'black',
points.col.border = 'black',
points.cex = 1,
add.line.segments = FALSE,
line.start = NULL,
line.end = NULL,
line.col = 'black',
```

```
line.lwd = 1,
add.text = FALSE,
text.labels = NULL,
text.x = NULL,
text.y = NULL,
text.col = 'black',
text.cex = 1,
text.fontface = 'bold',
text.guess.labels = FALSE,
text.guess.skip.labels = TRUE,
text.guess.ignore.radius = FALSE,
text.guess.ignore.rectangle = FALSE,
text.guess.radius.factor = 1,
text.guess.buffer.factor = 1,
text.guess.label.position = NULL,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
group.specific.colouring = TRUE,
use.legacy.settings = FALSE,
inside.legend.auto = FALSE,
regions.labels = c(),
        regions.start = c(),
regions.stop = c(),
regions.color = c("red"),
regions.cex = 1,
regions.alpha = 1,
        lollipop.bar.y = NULL,
lollipop.bar.color = "gray",
...
);
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame. Transforming data within formula is not compatible with automatic scaling with ‘xat‘ or ‘yat‘.
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`groups`	The grouping variable in the data-frame
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered

`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title
`xlab.label`	x-axis label
`ylab.label`	y-axis label
`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Accepts a vector listing where x-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes x-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`yat`	Accepts a vector listing where y-axis ticks should be drawn or if automatic scaling is desired, one of three strings: “auto”, “auto.linear” or “auto.log”. Automatic scaling fixes y-axis tick locations, labels, and data values dependent given data. “auto” will determine whether linear or logarithmic scaling fits the given data best, “auto.linear” or “auto.log” will force data to be scaled linearly or logarithmically respectively. Defaults to lattice automatic (TRUE). For more details see ‘auto.axis()‘.
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with xat will overwrite user input. Set to NULL to remove x-axis labels.
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic (TRUE). Using automatic scaling with yat will overwrite user input. Set to NULL to remove y-axis labels.
`xaxis.log`	Logical indicating whether x-variable should be in logarithmic scale (and what base if numeric)
`yaxis.log`	Logical indicating whether y-variable should be in logarithmic scale (and what base if numeric)
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2

`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`yaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`add.grid`	Logical stating wheter or not the grid should be drawn on the plot
`xgrid.at`	Vector listing where the x-axis grid lines should be drawn, defaults to xat
`ygrid.at`	Vector listing where the y-axis grid lines should be drawn, defaults to yat
`grid.colour`	ability to set individual grid line colours
`horizontal`	xyplot-specific function that allows you to change if type=’h’ draws lines to the vertical or horizontal axis
`type`	Accepts character vector of one or more elements defining how x and y are to be plotted. Accepted elements include: "p" to draw points, "l" to connect points with lines, "h" to draw vertical or horizonal line segments from the points to the origin, "s" or "S" to plot as a step curve, "g" to add a grid, and "r" to add a linear regression line. For more options and detail see "type" parameter in "xyplot" documentation.
`cex`	Character expansion for plotting symbol
`pch`	Plotting character
`col`	Point/line colour
`col.border`	Colour of border when points pch >= 21. Defaults to “black”
`lwd`	Specifies line width, defaults to 1
`lty`	Specifies line style, defaults to 1 (solid)
`alpha`	Specifies line transparency, defaults to 1 (opaque)
`axes.lwd`	Thickness of width of axes lines
`strip.col`	Strip background colour, defaults to “white”
`strip.cex`	Strip title character expansion
`strip.fontface`	Strip title fontface, defaults to bold
`y.error.up`	upward error vector. Defaults to NULL. When y.error.up is NULL, vertical error bar is not drawn
`y.error.down`	Downward error vector. Defaults to y.error.down to show symmetric error bars
`x.error.right`	Rightward error vector. Defaults to NULL. When x.error.right is NULL, horizontal error bar is not drawn
`x.error.left`	Leftward error vector. Defaults to x.error.right to show symmetric error bars
`y.error.bar.col`	Colour of vertical error bar. Defaults to “black”

`x.error.bar.col`	Colour of horizontal error bar. Defaults to “black”
`error.whisker.angle`	Angle of the whisker drawn on error bar. Defaults to 90 degree
`error.bar.lwd`	Error bar line width. Defaults to 1
`error.bar.length`	Length of the error bar whiskers. Defaults to 0.1
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.1
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`key.top`	A number specifying the distance at top of key, defaults to 0.1
`key.left.padding`	Amount of padding to go onto any legend on the left
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 1
`axis.key.padding`	A number specifying the distance from the y-axis to the key, defaults to 1
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”
`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`add.axes`	Allow axis lines to be turned on or off, default is FALSE
`axes.lty`	Specifies axis line style, defaults to “dashed”
`add.xyline`	Allow y=x line to be drawn, default is FALSE
`xyline.col`	y=x line colour, defaults to black
`xyline.lwd`	Specifies y=x line width, defaults to 1
`xyline.lty`	Specifies y=x line style, defaults to 1 (solid)
`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.v`	Allow vertical line to be drawn, default to NULL
`abline.col`	Horizontal line colour, defaults to black
`abline.lwd`	Specifies horizontal line width, defaults to 1

`abline.lty`	Specifies horizontal line style, defaults to 1 (solid)
`add.curves`	Allow curves to drawn, default is FALSE
`curves.exprs`	A list of functions, expressions, or calls using “x” as a variable that specify the curves to be drawn
`curves.from`	Specifies the x co-ordinates at which the start of each curve should be drawn, defaults to drawing the curves to the left edge of the plotting region
`curves.to`	Specifies the x co-ordinates at which the end of each curve should be drawn, defaults to drawing the curves to the right edge of the plotting region
`curves.col`	Specifies colours of curves, default is black for each curve
`curves.lwd`	Specifies width of curves, default is 1 for each curve
`curves.lty`	Specifies type of curves, default is 1 (solid) for each curve
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x ooordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`add.points`	Allow additional points to be drawn, default is FALSE
`points.x`	The x co-ordinates where additional points should be drawn
`points.y`	The y co-ordinates where additional points should be drawn
`points.pch`	The plotting character for additional points
`points.col`	The colour of additional points
`points.col.border`	Colour of the border of additional points if points.pch >= 21. Defaults to black
`points.cex`	The size of additional points
`add.line.segments`	Allow additional line segments to be drawn, default is FALSE
`line.start`	The y co-ordinates where additional line segments should start
`line.end`	The y co-ordinates where additional line segments should end
`line.col`	The colour of additional line segments, default is black
`line.lwd`	The line width of additional line segments, default is 1
`add.text`	Allow additional text to be drawn, default is FALSE
`text.labels`	Labels for additional text
`text.x`	The x co-ordinates where additional text should be placed
`text.y`	The y co-ordinates where additional text should be placed

`text.col`	The colour of additional text
`text.cex`	The size of additional text
`text.fontface`	The fontface for additional text
`text.guess.labels`	Allows automatic labeling by considering values in text.x and text.y as a data point to be labelled, default is FALSE
`text.guess.skip.labels`	Provides an option to disregard automatic labelling algorithm if no space is available around a data point, thus forcing labelling if a collision is likely, default is TRUE
`text.guess.ignore.radius`	Allows the automatic labeling algorithm to ignore the radius space of a data point, useful to label a cluster of data points with a single text box, default is FALSE
`text.guess.ignore.rectangle`	Allows the atuomatic labeling algorithm to ignore the rectangle space of multiple potential label positions, default is FALSE
`text.guess.radius.factor`	A numeric value to factor the radius value to alter distance from the label and the data point
`text.guess.buffer.factor`	A numeric value to factor the buffer value to alter the space which is used to consider if data.points are potentially going to collide
`text.guess.label.position`	A numeric value between 0 and 360 to specify the percise angle of a text box center and the positive x-axis. Angles move counter-clockwise beginning at the positive x axis
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`group.specific.colouring`	Variable to specify if group specific multi colouring for error bars is enforced
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function

`regions.labels`	Labels for each of the regions on the lollipop plots bars
`regions.start`	start x value of each of the regions
`regions.stop`	stop value for each of the regions
`regions.color`	color of each of the regions
`regions.cex`	size of the text of each of the regions
`regions.alpha`	alpha of each of the regions
`lollipop.bar.y`	y location of top of the lollipop plot bar – defaults to right above the bottom y axis
`lollipop.bar.color`	color of the lollipop plot bar
`...`	Additional arguments to be passed to xyplot

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`xyplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = rnorm(800),
    y = rnorm(800)
    );
create.scatterplot(
#    # filename = tempfile(pattern = 'Scatterplot_Simple', fileext = '.tiff'),
```

```
        formula = y ~ x,
        data = simple.data,
        resolution = 50
        );

    scatter.data <- data.frame(
        sample.one = microarray[1:800,1],
        sample.two = microarray[1:800,2],
        chr = microarray$Chr[1:800]
        );

    # Minimal Input
    create.scatterplot(
        # filename = tempfile(pattern = 'Scatterplot_Minimal_Input', fileext = '.tiff'),
        formula = sample.two ~ sample.one,
        data = scatter.data,
        main = 'Minimal Input',
        description = 'Scatter plot created by BoutrosLab.plotting.general',
        resolution = 50
        );

    # Axes & Labels
    create.scatterplot(
        # filename = tempfile(pattern = 'Scatterplot_Axes_Labels', fileext = '.tiff'),
        formula = sample.two ~ sample.one,
        data = scatter.data,
        main = 'Axes & Labels',
        # Axes and labels
        xlab.label = colnames(microarray[1]),
        ylab.label = colnames(microarray[2]),
        xat = seq(0, 16, 2),
        yat = seq(0, 16, 2),
        xlimits = c(0, 15),
        ylimits = c(0, 15),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        description = 'Scatter plot created by BoutrosLab.plotting.general',
        resolution = 50
        );

    # Log-Scaled Axis
    log.data <- data.frame(
        x = rnorm(800),
        y = 10 ** rnorm(800, mean = 5, sd = 2)
        );

    create.scatterplot(
        formula = y ~ x,
        data = log.data,
```

```
    # Log base 10 scale y-axis
    yat = 'auto.log',
    main = 'Log Scaled',
    description = 'Scatter created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Colour & Plotting Character
create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Colour_Pch', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Colour & Pch',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # setting the colour
    col = default.colours(2)[2],
    # setting the plotting character type & size
    pch = 21,
    cex = 1.5,
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colour depth
# create colour scheme to illustrate adding a colourkey
chr.palette <- colour.gradient(default.colours(2)[2], 800);

create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Colour_Depth', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Colour Depth',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
```

```
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # setting the colour
    col = chr.palette,
    # setting the plotting character type & size
    pch = 19,
    cex = 1,
    # adding key for colours
    key.top = 1.5,
    legend = list(
        bottom = list(
            fun = draw.colorkey,
            args = list(
                key = list(
                    col = chr.palette,
                    at = 1:800,
                    tick.number = 3,
                    space = 'bottom',
                    size = 1,
                    width = 1.25,
                    height = 1,
                    labels = list(
                        labels = 1:3,
                        cex = 1,
                at = c(1, which(scatter.data$chr == 2)[1], which(scatter.data$chr == 3)[1])
                        )
                    )
                )
            )
        ),
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Groups & Legend
create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Groups_Legend', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Groups & Legend',
    # using arbitrary groups for the sake of illustration
    groups = scatter.data$chr,
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
```

```
        ylab.cex = 1.5,
        col = default.colours(3),
        # Adding legend for groups
        key = list(
            text = list(
                lab = c('1','2','3'),
                cex = 1,
                col = 'black'
                ),
            points = list(
                pch = 19,
                col = default.colours(3),
                cex = 1
                ),
            x = 0.04,
            y = 0.95,
            padding.text = 2
            ),
        description = 'Scatter plot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Correlation Key
    create.scatterplot(
        # filename = tempfile(pattern = 'Scatterplot_Correlation_Key', fileext = '.tiff'),
        formula = sample.two ~ sample.one,
        data = scatter.data,
        main = 'Correlation Key',
        xlab.label = colnames(microarray[1]),
        ylab.label = colnames(microarray[2]),
        xat = seq(0, 16, 2),
        yat = seq(0, 16, 2),
        xlimits = c(0, 15),
        ylimits = c(0, 15),
        xaxis.cex = 1,
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        col = 'black',
        pch = 21,
        # Adding correlation key
        legend = list(
            inside = list(
                fun = draw.key,
                args = list(
                    key = get.corr.key(
                        x = scatter.data$sample.one,
                        y = scatter.data$sample.two,
                        label.items = c('spearman','spearman.p','kendall','beta1'),
                        alpha.background = 0,
                        key.cex = 1
```

```
                    )
                ),
            x = 0.04,
            y = 0.95,
            corner = c(0,1)
            )
        ),
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Panel Organization
create.scatterplot(
   # filename = tempfile(pattern = 'Scatterplot_Panel_numeric_conditional', fileext = '.tiff'),
    formula = sample.two ~ sample.one | chr,
    data = scatter.data,
    main = 'Panel',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    # set up panel layout
    layout = c(1,3),
    yrelation = 'free',
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 100
    );

scatter.data$chromosome <- as.character(scatter.data$chr);

create.scatterplot(
   # filename = tempfile(pattern = 'Scatterplot_Panel_character_conditional', fileext = '.tiff'),
    formula = sample.two ~ sample.one | chromosome,
    data = scatter.data,
    main = 'Panel',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
```

```
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    # set up panel layout
    layout = c(1,3),
    yrelation = 'free',
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Covariates
cov.colours <- as.character(microarray$Chr[1:800]);
cov.colours[cov.colours == '1'] <- default.colours(3, palette.type = 'chromosomes')[1];
cov.colours[cov.colours == '2'] <- default.colours(3, palette.type = 'chromosomes')[2];
cov.colours[cov.colours == '3'] <- default.colours(3, palette.type = 'chromosomes')[3];

cov <- list(
    rect = list(
        col = 'transparent',
        fill = cov.colours
        )
    );

cov.grob <- covariates.grob(
    covariates = cov,
    ord = c(1:length(cov.colours)),
    side = 'top',
    size = 1
    );

cov.legend <- list(
    legend = list(
        colours = default.colours(3, palette.type = 'chromosomes'),
        labels = c('1', '2', '3'),
        title = 'Chromosome',
        border = 'transparent'
        )
    );

cov.legend.grob <- legend.grob(
    legends = cov.legend
    );

create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Covariates', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
```

```
    main = 'Covariates',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    # Adding covariate & legend
    legend = list(
        bottom = list(fun = cov.grob),
        right = list(fun = cov.legend.grob)
        ),
    # Ensuring sufficient spacing for covariate
    key.top = 3,
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Error bars
error.data <- data.frame(
    chr = (microarray$Start)[1:20],
    values = apply(microarray[1:20,1:58], 1, mean),
    error = apply(microarray[1:20,1:58], 1, sd)
    );

create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Error_Bars', fileext = '.tiff'),
    formula = values ~ chr,
    data = error.data,
    main = 'Error Bars',
    xlab.label = 'Base pair location on chromosome one',
    ylab.label = 'Gene expression change',
    # xat = seq(0, 16, 2),
    yat = seq(0, 14, 2),
    # xlimits = c(0, 15),
    ylimits = c(0, 13),
    # Format xaxes
    xaxis.lab = c(
        scientific.notation(0, 1),
        scientific.notation(1000000, 1),
        scientific.notation(2000000, 1),
        scientific.notation(3000000, 1),
        scientific.notation(4000000, 1),
        scientific.notation(5000000, 1),
```

```
        scientific.notation(6000000, 1),
        scientific.notation(7000000, 1)
        ),
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 19,
    col = 'black',
    fill = 'transparent',
    # Specifying error bars
    error.bar.lwd = 1,
    error.whisker.angle = 120,
    y.error.up = error.data$error,
    y.error.bar.col = 'black',
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );

create.scatterplot(
   # filename = tempfile(pattern = 'Scatterplot_Error_Bars_MultiColor', fileext = '.tiff'),
    formula = values ~ chr,
    data = error.data,
    main = 'Error Bars',
    xlab.label = 'Base pair location on chromosome one',
    ylab.label = 'Gene expression change',
    # xat = seq(0, 16, 2),
    yat = seq(0, 14, 2),
    # xlimits = c(0, 15),
    ylimits = c(0, 13),
    # Format xaxes
    xaxis.lab = c(
        scientific.notation(0, 1),
        scientific.notation(1000000, 1),
        scientific.notation(2000000, 1),
        scientific.notation(3000000, 1),
        scientific.notation(4000000, 1),
        scientific.notation(5000000, 1),
        scientific.notation(6000000, 1),
        scientific.notation(7000000, 1)
        ),
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 19,
    col = 'black',
```

```
    fill = 'transparent',
    # Specifying error bars
    error.bar.lwd = 1,
    error.whisker.angle = 120,
    y.error.up = error.data$error,
    y.error.bar.col = c('black','red','blue'),
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    group.specific.colouring = FALSE,
    resolution = 200
    );


# Gridlines
create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Gridlines', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Gridlines',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    # Adding gridlines
    type = c('p','g'),
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# lines & background rectangle
create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Lines_BG', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Lines & BG rectangle',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
```

```
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    type = c('p','g'),
    # add xy line
    add.xyline = TRUE,
    xyline.lty = 3,
    xyline.col = 'red',
    xyline.lwd = 3,
    # add background rectangle
    add.rectangle = TRUE,
    xleft.rectangle = which(scatter.data$chr == 2)[1]/800*15,
    xright.rectangle = which(scatter.data$chr == 3)[1]/800*15,
    ybottom.rectangle = 0,
    ytop.rectangle = 15,
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# attach lines to points
create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Lines', fileext = '.tiff'),
    formula = sample.two ~ sample.one | chr,
    data = scatter.data,
    main = 'Lines',
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    pch = 21,
    col = 'black',
    fill = 'transparent',
    # attach lines
    type = c('h','p'),
    layout = c(1,3),
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

```
# ROC curve
set.seed(123456);

class.values <- runif(50, 0, 1);
observed.values <- sample(c(0,1), size = 50, replace = TRUE);
cutoffs <- seq(1,0,-0.01);
tprs <- c();
fprs <- c();

for (c in cutoffs) {
        roc.classification <- rep(0, length(class.values));
        roc.classification[class.values >= c] <- 1;
        roc.results <- table(
                factor(roc.classification, levels = c(0,1)),
                factor(observed.values, levels = c(0,1)),
                dnn = c('pred', 'obs')
                );
        tprs <- c(tprs, roc.results[2,2] / (roc.results[2,2] + roc.results[1,2]));
        fprs <- c(fprs, roc.results[2,1] / (roc.results[2,1] + roc.results[1,1]));
        }

roc.data <- data.frame(cutoff = cutoffs, TPR = tprs, FPR = fprs);
points.x <- roc.data[match(c(0.25, 0.5, 0.75), roc.data$cutoff), 'FPR'];
points.y <- roc.data[match(c(0.25, 0.5, 0.75), roc.data$cutoff), 'TPR'];

create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_ROC', fileext = '.tiff'),
    formula = TPR ~ FPR,
    data = roc.data,
    main = 'ROC',
    xlab.label = 'False positive rate',
    ylab.label = 'True positive rate',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # To plot ROC curve, add "s" or "S" to type vector.
    # "s" connects points with the vertical segment first.
    # "S" connects points with the horizontal segment first.
    type = 's',
    lwd = 3,
    add.xyline = TRUE,
    xyline.col = 'grey',
    add.points = TRUE,
    points.x = points.x,
    points.y = points.y,
    points.col = c('blue', 'darkgreen', 'red'),
    add.text = TRUE,
    text.labels = paste('cutoff = ', c(0.25, 0.5, 0.75), sep = ''),
    #text.x = points.x - 0.14,
    #text.y = points.y + 0.03,
```

```
    text.x = points.x,
    text.y = points.y,
    text.guess.labels = TRUE,
    text.guess.label.position = 155,
    text.guess.radius.factor = 2.5,
    description = 'Scatter plot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Volcano Plots
fold.change <- apply(microarray[,1:29], 1, mean) - apply(microarray[,30:58], 1, mean);

fake.microarray <- microarray[,1:58] - log(mean(apply(microarray[,1:58],1, mean)));
fake.microarray[,30:58] <-  fake.microarray[,30:58] + mean(fold.change);
fake.microarray[fake.microarray < 0] <- 0;

p.values <- apply(fake.microarray[,1:58], 1, function(x) {t.test(x=x[1:29],y=x[30:58])$p.value} );
fold.change <- apply(fake.microarray[, 1:29], 1, mean) - apply(fake.microarray[, 30:58], 1,mean);
p.values.adjusted <- p.adjust(p.values, 'fdr');

dot.colours <- vector(length=length(p.values));
dot.colours[p.values.adjusted < .05 & fold.change < 0] <- 'green';
dot.colours[p.values.adjusted < .05 & fold.change > 0] <- 'red';
dot.colours[p.values.adjusted > .05] <- 'black';


volcano.data <- data.frame(
    p.values = -log10(p.values.adjusted),
    fold.change = fold.change
    );

create.scatterplot(
    # filename = tempfile(pattern = 'Scatterplot_Volcano_Plot', fileext = '.tiff'),
    formula = p.values ~ fold.change,
    data = volcano.data,
    col = dot.colours,
    alpha = .5,
    yat = c(0,2,4,6,8),
    ylimits = c(-0.1,8.1),
    yaxis.lab = expression(10^0,10^-2,10^-4,10^-6,10^-8),
    yaxis.cex = 1.5,
    xaxis.cex = 1.5,
    xlab.label = 'foldChange',
    ylab.label = 'pValues',
    xlab.cex = 1.75,
    ylab.cex = 1.75,
    resolution = 200
    );

# Automatic Labeling
interesting.fold.change <- (fold.change < -.9 | fold.change > .9);
interesting.p.value <- (-log10(p.values.adjusted) < 8 & -log10(p.values.adjusted) > 2);
interesting.points <- interesting.fold.change & interesting.p.value;
```

```
text.x <- fold.change[interesting.points];
text.y <- (-log10(p.values.adjusted))[interesting.points];
text.labels <- rownames(microarray)[interesting.points];

create.scatterplot(
   # filename = tempfile(pattern = 'Scatterplot_Volcano_Plot_With_Labels', fileext = '.tiff'),
    formula = p.values ~ fold.change,
    data = volcano.data,
    alpha = .5,
    yat = c(0,2,4,6,8),
    ylimits = c(-0.1,8.1),
    xlimits = c(-1.5,1.5),
    yaxis.lab = expression(10^0,10^-2,10^-4,10^-6,10^-8),
    yaxis.cex = 1.5,
    xaxis.cex = 1.5,
    xlab.label = 'foldChange',
    ylab.label = 'pValues',
    xlab.cex = 1.75,
    ylab.cex = 1.75,
    add.text = TRUE,
    text.x = text.x,
    text.y = text.y,
    text.labels = text.labels,
    text.guess.labels = TRUE,
    resolution = 200
    );

# With line segments
line.data <- data.frame(
    group = as.factor(c('A','B','C')),
    x = sample(1:10,3),
    y = sample(1:10,3),
    z = sample(1:10,3)
    );

create.scatterplot(
    (x+y+z) ~ group,
    line.data,
    # filename = tempfile(pattern = 'Scatterplot_with_LineSegments', fileext = '.tiff'),
    cex = 0,
    add.line.segments = TRUE,
    line.start = list(
rep(0,nrow(line.data)),
line.data$x,
c(line.data$x + line.data$y)
),
    line.end = list(
line.data$x,
c(line.data$x + line.data$y),
c(line.data$x + line.data$y + line.data$z)
),
    line.col = list('red','blue','green'),
```

```
        line.lwd = list(3,3,3),
        resolution = 200
        );

    lollipop.data <- data.frame(
        y = seq(1,100,1),
        x = rnorm(100)
        );

    create.lollipopplot(
        # filename = tempfile(pattern = 'Lollipop_Simple', fileext = '.tiff'),
        formula = x ~ y,
        data = lollipop.data,
        main = 'Lollipop plot',
        xaxis.cex = 1,
        xlimits = c(-1,102),
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        pch = 21,
        col = 'black',
        fill = 'transparent',
        description = 'Scatter plot created by BoutrosLab.plotting.general',
        regions.start = c(1,26,48),
        regions.stop = c(15,35,72),
        regions.labels = c("test 1", "test2", "test 3"),
        regions.color = c("#66b3ff", "#5cd65c", "#ff3333"),
        resolution = 200
        );
```

`create.segplot` *Make a segplot*

Description

Takes a data.frame and creates a segplot

Usage

```
create.segplot(
formula,
data,
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
```

```
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
xaxis.lab = TRUE,
yaxis.lab = TRUE,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.tck = 1,
yaxis.tck = 1,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
abline.h = NULL,
abline.v = NULL,
abline.lty = 1,
abline.lwd = 1,
abline.col = 'black',
segments.col = 'black',
segments.lwd = 1,
layout = NULL,
as.table = FALSE,
x.spacing = 0,
y.spacing = 0,
x.relation = 'same',
y.relation = 'same',
top.padding = 0.5,
bottom.padding = 2,
right.padding = 1,
left.padding = 2,
ylab.axis.padding = 0,
level = NULL,
```

```
col.regions = NULL,
centers = NULL,
plot.horizontal = TRUE,
draw.bands = FALSE,
pch = 16,
symbol.col = 'black',
symbol.cex = 1,
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
axes.lwd = 1,
key = NULL,
legend = NULL,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
inside.legend.auto = FALSE,
disable.factor.sorting = FALSE
)
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 3
`xlab.label`	x-axis label
`ylab.label`	y-axis label
`xlab.cex`	Size of x-axis label, defaults to 2
`ylab.cex`	Size of y-axis label, defaults to 2
`xlab.col`	Colour of the x-axis label, defaults to “black”

`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis scales, defaults to 1.5
`yaxis.cex`	Size of y-axis scales, defaults to 1.5
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.fontface`	Fontface for the x-axis scales, defaults to “plain”
`yaxis.fontface`	Fontface for the y-axis scales, defaults to “plain”
`xaxis.rot`	Counterclockwise rotation of text in x-axis scales in degrees, defaults to 0
`yaxis.rot`	Counterclockwise rotation of text in y-axis scales in degrees, defaults to 0
`xaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`yaxis.tck`	Specifies the length of the tick mark, defaults to 1 for both top and bottom axes
`xlimits`	Two-element vector giving the x-axis limits, defaults to automatic
`ylimits`	Two-element vector giving the y-axis limits, defaults to automatic
`xat`	Vector listing where the x-axis labels should be drawn, defaults to automatic
`yat`	Vector listing where the y-axis labels should be drawn, defaults to automatic
`abline.h`	Allow horizontal line to be drawn, default to NULL
`abline.v`	Allow vertical line to be drawn, default to NULL
`abline.lty`	Specifies horizontal line style, defaults to 1 (solid)
`abline.lwd`	Specifies horizontal line width, defaults to 1
`abline.col`	Horizontal line colour, defaults to black
`segments.col`	Colour of segments, defaults to “black”
`segments.lwd`	Line width of segments, defaults to 1
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
`as.table`	Specifies panel drawing order, default is FALSE which draws panels from bottom left corner, moving right then up. Set to TRUE to draw from top left corner, moving right then down
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`x.relation`	Allows x-axis scales to vary if set to “free”, defaults to “same”

`y.relation`	Allows y-axis scales to vary if set to “free”, defaults to “same”
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.1
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 1
,	
`level`	Optional covariate that determines colour coding of the segments, if specified overwrites segments.col, can contain actual colors or values to determine colors, then col.regions should be defined
`col.regions`	Vector of colors, define if level is numeric
`centers`	Optional vector for centers of segments, defaults to NULL
`plot.horizontal`	Logical whether segments should be drawn horizontally (default) or vertically
`draw.bands`	Logical to specify whether to draw lines (default) or rectangles
`pch`	Plotting character for centers
`symbol.col`	Colour of plotting character for centers, defaults to “black”
`symbol.cex`	Size of plotting character for centers, defaults to 1
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle’s area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`axes.lwd`	Specifies axes line width, defaults to 1
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE

`description`	Short description of image/plot; default NULL
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function
`disable.factor.sorting`	Disable barplot auto sorting factors alphabetically/numerically

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`levelplot`, `segplot` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    min = runif(10,5,15),
    max = runif(10,15,25),
    labels = as.factor(LETTERS[1:10])
    );

create.segplot(
```

```
    # filename = tempfile(pattern = 'Segplot_simple', fileext = '.tiff'),
    formula = labels ~ min + max,
    data = simple.data,
    resolution = 50
    );

# load some data
length.of.gene <- apply(microarray[1:10,60:61], 1, diff);
bin.length <- length.of.gene;
bin.length[which(bin.length < 20000)] <- 'A';
bin.length[which(bin.length < 40000)] <- 'B';
bin.length[which(bin.length < 60000)] <- 'C';

segplot.data <- data.frame(
    min = apply(microarray[1:10,1:58], 1, min),
    max = apply(microarray[1:10,1:58], 1, max),
    median = apply(microarray[1:10,1:58], 1, median),
    gene = as.factor(rownames(microarray)[1:10]),
    # approximating length of gene
    length = as.factor(bin.length)
    );

# Minimal Input using real data
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Minimal_Input', fileext = '.tiff'),
    formula = gene ~ min + max,
    data = segplot.data,
    main = 'Minimal input',
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes & Labels
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Axes_Labels', fileext = '.tiff'),
    formula = gene ~ min + max,
    data = segplot.data,
    main = 'Axes & labels',
    # Formatting axes
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,13),
    xat = seq(0, 12, 2),
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Bands
```

```
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Bands', fileext = '.tiff'),
    formula = gene ~ min + max,
    data = segplot.data,
    main = 'Bands',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,13),
    xat = seq(0, 12, 2),
    # drawing rectangles instead of lines
    draw.bands = TRUE,
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colours
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Colours', fileext = '.tiff'),
    formula = reorder(gene, median) ~ min + max,
    data = segplot.data,
    main = 'Colours',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,13),
    xat = seq(0, 12, 2),
    draw.bands = FALSE,
    # Changing the colours based on a covariate ('level' parameter)
    level = segplot.data$length,
    col.regions = default.colours(3),
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Median
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Median', fileext = '.tiff'),
    formula = gene ~ min + max,
    data = segplot.data,
    main = 'Medians',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
```

```
        xlimits = c(0,13),
        draw.bands = FALSE,
        xat = seq(0, 12, 2),
        level = segplot.data$length,
        col.regions = default.colours(3),
        # Adding center values
        centers = segplot.data$median,
        description = 'Segplot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Reorder by center value
    create.segplot(
        # filename = tempfile(pattern = 'Segplot_Reorder', fileext = '.tiff'),
        formula = reorder(gene, median) ~ min + max,
        data = segplot.data,
        main = 'Reordered',
        xlab.label = 'Change in gene expression',
        ylab.label = 'Gene',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xlimits = c(0,13),
        xat = seq(0, 12, 2),
        draw.bands = FALSE,
        centers = segplot.data$median,
        level = segplot.data$length,
        col.regions = default.colours(3),
        description = 'Segplot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # Legend
    create.segplot(
        # filename = tempfile(pattern = 'Segplot_Legend', fileext = '.tiff'),
        formula = reorder(gene, median) ~ min + max,
        data = segplot.data,
        main = 'Legend',
        xlab.label = 'Change in gene expression',
        ylab.label = 'Gene',
        xaxis.cex = 1,
        yaxis.cex = 1,
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        xlimits = c(0,13),
        xat = seq(0, 12, 2),
        draw.bands = FALSE,
        centers = segplot.data$median,
        level = segplot.data$length,
        col.regions = default.colours(3),
        # Adding legend to explain colours
        legend = list(
```

```
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 19,
                        cex = 1
                        ),
                    text = list(
                        lab = c('1-20000 bp', '20001-40000 bp','40001-60000 bp')
                        ),
                    padding.text = 1,
                    cex = 1
                    )
                ),
            x = 0.60,
            y = 0.15,
            corner = c(0,1)
            )
        ),
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Background
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Background', fileext = '.tiff'),
    formula = reorder(gene, median) ~ min + max,
    data = segplot.data,
    main = 'Background rectangle',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,13),
    xat = seq(0, 12, 2),
    draw.bands = FALSE,
    centers = segplot.data$median,
    level = segplot.data$length,
    col.regions = default.colours(3),
    # Adding legend to explain colours
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 19,
                        cex = 1
```

```
                        ),
                    text = list(
                        lab = c('1-20000 bp', '20001-40000 bp','40001-60000 bp')
                        ),
                    padding.text = 1,
                    cex = 1
                    )
                ),
            x = 0.50,
            y = 0.15,
            corner = c(0,1)
            )
        ),
    # adding background shading
    add.rectangle = TRUE,
    xleft.rectangle = 0,
    ybottom.rectangle = seq(0.5, 8.5, 2),
    xright.rectangle = 13,
    ytop.rectangle = seq(1.5, 9.5, 2),
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,
    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Nature style
create.segplot(
    # filename = tempfile(pattern = 'Segplot_Nature_style', fileext = '.tiff'),
    formula = reorder(gene, median) ~ min + max,
    data = segplot.data,
    main = 'Nature style',
    xaxis.cex = 1,
    yaxis.cex = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xlimits = c(0,13),
    xat = seq(0, 12, 2),
    draw.bands = FALSE,
    centers = segplot.data$median,
    level = segplot.data$length,
    col.regions = default.colours(3),
    legend = list(
        inside = list(
            fun = draw.key,
            args = list(
                key = list(
                    points = list(
                        col = default.colours(3),
                        pch = 19,
                        cex = 1
                        ),
                    text = list(
                        lab = c('1-20000 bp', '20001-40000 bp','40001-60000 bp')
```

```
                        ),
                    padding.text = 1,
                    cex = 1
                    )
                ),
            x = 0.50,
            y = 0.15,
            corner = c(0,1)
            )
        ),
    add.rectangle = TRUE,
    xleft.rectangle = 0,
    ybottom.rectangle = seq(0.5, 8.5, 2),
    xright.rectangle = 13,
    ytop.rectangle = seq(1.5, 9.5, 2),
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Segplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# example of bands and lines
create.segplot(
    # filename = tempfile(pattern = 'Segplot_BandsAndLines', fileext = '.tiff'),
    formula = labels ~ min + max,
    data = simple.data,
    draw.bands = c(1,3,5,7,9),
    resolution = 200
    );
```

`create.stripplot` *Make a strip-plot*

Description

Takes a formula and a data.frame and creates a strip-plot

Usage

```
create.stripplot(
formula,
data,
filename = NULL,
groups = NULL,
jitter.data = FALSE,
jitter.factor = 1,
jitter.amount = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
xlab.top.cex = 2,
xlab.top.col = 'black',
xlab.top.just = 'center',
xlab.top.x = 0.5,
xlab.top.y = 0,
xaxis.lab = TRUE,
yaxis.lab = TRUE,
xaxis.cex = 1.5,
yaxis.cex = 1.5,
xaxis.col = 'black',
yaxis.col = 'black',
xaxis.fontface = 'bold',
yaxis.fontface = 'bold',
xaxis.rot = 0,
yaxis.rot = 0,
xaxis.tck = 0,
yaxis.tck = 1,
xlimits = NULL,
ylimits = NULL,
xat = TRUE,
yat = TRUE,
lwd = 1,
pch = 19,
col = 'black',
col.border = 'black',
fill = 'transparent',
colour.alpha = 1,
```

```
cex = 0.75,
top.padding = 0.1,
bottom.padding = 0.7,
right.padding = 0.3,
left.padding = 0.5,
ylab.axis.padding = 1,
layout = NULL,
as.table = TRUE,
x.spacing = 0,
y.spacing = 0,
add.median = FALSE,
median.values = NULL,
add.rectangle = FALSE,
xleft.rectangle = NULL,
ybottom.rectangle = NULL,
xright.rectangle = NULL,
ytop.rectangle = NULL,
col.rectangle = 'transparent',
alpha.rectangle = 1,
strip.col = 'white',
strip.cex = 1,
strip.fontface = 'bold',
key = NULL,
legend = NULL,
height = 6,
width = 6,
size.units = 'in',
resolution = 1600,
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
        inside.legend.auto = FALSE,
disable.factor.sorting = FALSE
)
```

Arguments

`formula`	The formula used to extract the x & y components from the data-frame
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`groups`	The grouping variable in the data-frame
`jitter.data`	Allow data to be staggered, default is FALSE
`jitter.factor`	Numeric value to apply to jitter, default is 1
`jitter.amount`	Numeric; amount of noise to add, default is NULL

`main`	The main title for the plot (space is reclaimed if NULL)
s	
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title
`xlab.label`	X-axis label
`ylab.label`	Y-axis label
`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis scales, defaults to 2
`yaxis.cex`	Size of y-axis scales, defaults to 2
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales
`xaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to 0
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to 1
`xlimits`	Two-element vector giving the x-axis limits, default is automatic
`ylimits`	Two-element vector giving the y-axis limits, default is automatic
`xat`	Vector listing where the x-axis labels should be drawn, default is automatic
`yat`	Vector listing where the y-axis labels should be drawn, default is automatic
`lwd`	Line width, defaults to 1
`pch`	The plotting character (defaults to filled circles)
`col`	Colour of the plotting character (defaults to black)

`col.border`	Colour of border when pch > 21. Defaults to black
`fill`	Fill colour of the plotting character if pch set to 21:25 (defaults to transparent)
`colour.alpha`	Bias to be added to colour selection (defaults to 1)
`cex`	The size of the plotting character
`top.padding`	A number specifying the distance to the top margin, defaults to 0.1
`bottom.padding`	A number specifying the distance to the bottom margin, defaults to 0.7
`right.padding`	A number specifying the distance to the right margin, defaults to 0.3
`left.padding`	A number specifying the distance to the left margin, defaults to 0.5
`ylab.axis.padding`	A number specifying the distance of ylabel to the y-axis, defaults to 1
`layout`	A vector specifying the number of columns, rows (e.g., c(2,1). Default is NULL; see lattice::xyplot for more details
.	
`as.table`	Specifies panel drawing order, default is TRUE to draw from top left corner, moving right then down. Set to FALSE to draw panels from bottom left corner, moving right then up
`x.spacing`	A number specifying the distance between panels along the x-axis, defaults to 0
`y.spacing`	A number specifying the distance between panels along the y-axis, defaults to 0
`add.median`	TRUE/FALSE indicating whether lines should be drawn at the group medians, default is FALSE
`median.values`	A vector of values representing the median of each group, default is NULL
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`strip.col`	Strip background colour, defaults to "white"
`strip.cex`	Strip title character expansion
`strip.fontface`	Strip text fontface, defaults to bold
`key`	A list giving the key (legend). The default suppresses drawing
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`height`	Figure height, defaults to 6 inches

`width`	Figure width, defaults to 6 inches
`size.units`	Figure units, defaults to inches
`resolution`	Figure resolution in dpi, defaults to 1600
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`inside.legend.auto`	boolean specifying whether or not to use the automatic inside legend function
`disable.factor.sorting`	Disable barplot auto sorting factors alphabetically/numerically

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`stripplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = c(rep(rnorm(50),5)),
    y = as.factor(sample(LETTERS[1:5],250,TRUE))
    );

create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_simple', fileext = '.tiff'),
    formula = x ~ y,
    data = simple.data,
    resolution = 50
    );

# load real datasets
stripplot.data <- data.frame(
    values = c(t(microarray[1:10, 1:58])),
    genes = rep(rownames(microarray)[1:10], each = 58),
    sex = patient$sex,
    stringsAsFactors = TRUE
    );

# Minimal Input using real data
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_Minimal_Input', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'Minimal input',
    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 50
    );

# Axes & Labels
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_Axes_Labels', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'Axes & labels',
    # formatting axes
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 100
```

```
    );

# Colour & Legend
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_Colour_Legend', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'Colour & legend',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    # Colour & points adjustment
    groups = stripplot.data$sex,
    col = c('pink', 'skyblue'),
    pch = 19,
    colour.alpha = 0.5,
    cex = 1,
    # Legend
    key = list(
        space = 'right',
        text = list(
            lab = levels(stripplot.data$sex),
            cex = 1,
            col = 'black'
            ),
        points = list(
            pch = 19,
            col = c('pink','skyblue'),
            alpha = 0.5,
            cex = 1
            ),
        padding.text = 3
        ),
    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Jitter
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_Jitter', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'Low Jitter',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
```

```
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    groups = stripplot.data$sex,
    col = c('pink', 'skyblue'),
    pch = 19,
    colour.alpha = 0.5,
    cex = 1,
    key = list(
        space = 'right',
        text = list(
            lab = levels(stripplot.data$sex),
            cex = 1,
            col = 'black'
            ),
        points = list(
            pch = 19,
            col = c('pink','skyblue'),
            alpha = 0.4,
            cex = 1
            ),
        padding.text = 3
        ),
    # Custom jitter
    jitter.data = TRUE,
    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Jitter
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_High_Jitter', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'High Jitter',
    xlab.label = 'Change in gene expression',
    ylab.label = 'Gene',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    groups = stripplot.data$sex,
    col = c('pink', 'skyblue'),
    pch = 19,
```

```
    colour.alpha = 0.5,
    cex = 1,
    key = list(
        space = 'right',
        text = list(
            lab = levels(stripplot.data$sex),
            cex = 1,
            col = 'black'
            ),
        points = list(
            pch = 19,
            col = c('pink','skyblue'),
            alpha = 0.4,
            cex = 1
            ),
        padding.text = 3
        ),
    # Custom jitter
    jitter.data = TRUE,
    jitter.factor = 0.5,
    jitter.amount = 0.33,
    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 200
    );

# Nature style
create.stripplot(
    # filename = tempfile(pattern = 'Stripplot_Nature_style', fileext = '.tiff'),
    formula = genes ~ values,
    data = stripplot.data,
    main = 'Nature style',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlimits = c(0,13),
    xat = seq(0,12,2),
    groups = stripplot.data$sex,
    col = c('pink', 'skyblue'),
    pch = 19,
    colour.alpha = 0.5,
    cex = 1,
    key = list(
        space = 'right',
        text = list(
            lab = levels(stripplot.data$sex),
            cex = 1,
            col = 'black'
            ),
        points = list(
            pch = 19,
```

```
            col = c('pink','skyblue'),
            alpha = 0.4,
            cex = 1
            ),
        padding.text = 3
        ),
    jitter.data = TRUE,
    jitter.factor = 0.5,
    jitter.amount = 0.33,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.label = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.label = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Stripplot created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

`create.violinplot` *Make a violin plot*

Description

This function takes a dataframe and writes a pretty TIFF violin plot

Usage

```
create.violinplot(
formula,
data,
filename = NULL,
main = NULL,
main.just = 'center',
main.x = 0.5,
main.y = 0.5,
main.cex = 3,
xlab.label = tail(sub('~', '', formula[-2]), 1),
ylab.label = tail(sub('~', '', formula[-3]), 1),
xlab.cex = 2,
ylab.cex = 2,
xlab.col = 'black',
ylab.col = 'black',
xlab.top.label = NULL,
```

```
    xlab.top.cex = 2,
    xlab.top.col = 'black',
    xlab.top.just = 'center',
    xlab.top.x = 0.5,
    xlab.top.y = 0,
    xaxis.lab = TRUE,
    yaxis.lab = TRUE,
    xaxis.cex = 1.5,
    yaxis.cex = 1.5,
    xaxis.col = 'black',
    yaxis.col = 'black',
    xaxis.fontface = 'bold',
    yaxis.fontface = 'bold',
    xaxis.rot = 0,
    yaxis.rot = 0,
    xaxis.tck = c(1,0),
    yaxis.tck = c(1,1),
    ylimits = NULL,
    yat = TRUE,
    col = 'black',
    lwd = 1,
    border.lwd = 1,
    bandwidth = 'nrd0',
    bandwidth.adjust = 1,
    extra.points = NULL,
    extra.points.pch = 21,
    extra.points.col = 'white',
    extra.points.border = 'black',
    extra.points.cex = 1,
    start = NULL,
    end = NULL,
    scale = FALSE,
    plot.horizontal = FALSE,
    top.padding = 0.1,
    bottom.padding = 0.7,
    left.padding = 0.5,
    right.padding = 0.3,
    key = NULL,
    legend = NULL,
    add.rectangle = FALSE,
    xleft.rectangle = NULL,
    ybottom.rectangle = NULL,
    xright.rectangle = NULL,
    ytop.rectangle = NULL,
    col.rectangle = 'transparent',
    alpha.rectangle = 1,
    height = 6,
    width = 6,
```

```
resolution = 1600,
size.units = 'in',
enable.warnings = FALSE,
description = 'Created with BoutrosLab.plotting.general',
style = 'BoutrosLab',
preload.default = 'custom',
use.legacy.settings = FALSE,
disable.factor.sorting = FALSE
)
```

Arguments

`formula`	The formula used to extract the violin components from the data-frame
`data`	The data-frame to plot
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`main`	The main title for the plot (space is reclaimed if NULL)
`main.just`	The justification of the main title for the plot, default is centered
`main.x`	The x location of the main title, deault is 0.5
`main.y`	The y location of the main title, default is 0.5
`main.cex`	Size of text for main plot title, defaults to 3
`xlab.label`	The label for the x-axis
`ylab.label`	The label for the y-axis
`xlab.cex`	Size of x-axis label, defaults to 3
`ylab.cex`	Size of y-axis label, defaults to 3
`xlab.col`	Colour of the x-axis label, defaults to “black”
`ylab.col`	Colour of the y-axis label, defaults to “black”
`xlab.top.label`	The label for the top x-axis
`xlab.top.cex`	Size of top x-axis label
`xlab.top.col`	Colour of the top x-axis label
`xlab.top.just`	Justification of the top x-axis label, defaults to centered
`xlab.top.x`	The x location of the top x-axis label
`xlab.top.y`	The y location of the top y-axis label
`xaxis.lab`	Vector listing x-axis tick labels, defaults to automatic
`yaxis.lab`	Vector listing y-axis tick labels, defaults to automatic
`xaxis.cex`	Size of x-axis tick labels, defaults to 2
`yaxis.cex`	Size of y-axis tick labels, defaults to 2
`xaxis.col`	Colour of the x-axis tick labels, defaults to “black”
`yaxis.col`	Colour of the y-axis tick labels, defaults to “black”
`xaxis.fontface`	Fontface for the x-axis scales
`yaxis.fontface`	Fontface for the y-axis scales

`xaxis.rot`	Rotation of x-axis tick labels; defaults to 0
`yaxis.rot`	Rotation of y-axis tick labels; defaults to 0
`xaxis.tck`	Specifies the length of the tick marks for x-axis, defaults to c(1,0)
`yaxis.tck`	Specifies the length of the tick marks for y-axis, defaults to c(1,1)
`ylimits`	Two-element vector giving the y-axis limits, default is automatic
`yat`	Vector listing where the y-axis labels should be drawn, default is automatic
`col`	Colour to use for filling the interior of the violin plots, defaults to “black”
`lwd`	Line width, defaults to 1
`border.lwd`	Width of the exterior boundary of the violin plots, defaults to 1
`bandwidth`	Smoothing bandwidth, or character string giving rule to choose bandwidth (’nrd0’, ’nrd’, ’ucv’, ’bcv’, ’sj’, or ’sj-ste’). Passed to base R function density, via lattice::bwplot.
`bandwidth.adjust`	Adjustment parameter for the bandwidth (bandwidth used is bandwidth*bandwidth.adjust). Makes it easy to specify bandwidth as a proportion of the default.
`extra.points`	A list of numeric vectors, each one of length equal to the number of violins to be plotted. Specifies a set or sets of extra points to be plotted along the vertical spine of each violin plot. Defaults to NULL (no points to be added)
`extra.points.pch`	A vector of the same length as extra.points specifying the symbol to use for each set of points. Defaults to 21
`extra.points.col`	A vector of the same length as extra.points specifying the colour to use for each set of points. Defaults to “white”
`extra.points.border`	A vector of the same length as extra.points specifying the border colour to use for points >=21. Defaults to “black”
`extra.points.cex`	A vector of the same length as extra.points specifying the size of each set of points. Defaults to 1
`start`	Start of boundary cutoff, default is NULL for no boundary
`end`	End of boundary cutoff, default is NULL for no boundary
`scale`	Logical; Scales the violin plots, see ?panel.violin for more details, default is FALSE
`plot.horizontal`	Logical; Determines whether to draw violin plot horizontally or vertically; default is FALSE; If `horizontal` is `FALSE`, x will be coerced to a factor or shingle, and vise versa.
`top.padding`	A number giving the top padding in multiples of the lattice default
`bottom.padding`	A number giving the bottom padding in multiples of the lattice default
`left.padding`	A number giving the left padding in multiples of the lattice default
`right.padding`	A number giving the right padding in multiples of the lattice default

`key`	Add a key to the plot. See xyplot.
`legend`	Add a legend to the plot. Helpful for adding multiple keys and adding keys to the margins of the plot. See xyplot.
`add.rectangle`	Allow a rectangle to be drawn, default is FALSE
`xleft.rectangle`	Specifies the left x coordinate of the rectangle to be drawn
`ybottom.rectangle`	Specifies the bottom y coordinate of the rectangle to be drawn
`xright.rectangle`	Specifies the right x coordinate of the rectangle to be drawn
`ytop.rectangle`	Specifies the top y coordinate of the rectangle to be drawn
`col.rectangle`	Specifies the colour to fill the rectangle's area
`alpha.rectangle`	Specifies the colour bias of the rectangle to be drawn
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches
`resolution`	Figure resolution in dpi, defaults to 1600
`size.units`	Figure units, defaults to inches
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image/plot; default NULL
`style`	defaults to “BoutrosLab”, also accepts “Nature”, which changes parameters according to Nature formatting requirements
`preload.default`	ability to set multiple sets of diffrent defaults depending on publication needs
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`disable.factor.sorting`	Disable barplot auto sorting factors alphabetically/numerically

Value

If `filename` is `NULL` then returns the trellis object, otherwise creates a plot and returns a 0/1 success code.

Warning

If this function is called without capturing the return value, or specifying a filename, it may crash while trying to draw the histogram. In particular, if a script that uses such a call of create histogram is called by reading the script in from the command line, it will fail badly, with an error message about unavailable fonts:

```
Error in grid.Call.graphics("L_text", as.graphicsAnnot(x$label), x$x,  )
    Invalid font type
Calls: print ... drawDetails.text -> grid.Call.graphics -> .Call.graphics
```

Author(s)

Paul C. Boutros

See Also

`bwplot`, `lattice` or the Lattice book for an overview of the package.

Examples

```
set.seed(12345);

simple.data <- data.frame(
    x = c(rep(rnorm(50),5)),
    y = as.factor(sample(LETTERS[1:5],250,TRUE))
    );

create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Simple', fileext = '.tiff'),
    formula = x ~ y,
    data = simple.data,
    resolution = 100
    );

# load real datasets
violin.data <- data.frame(
    values = c(t(microarray[1:10, 1:58])),
    genes = rep(rownames(microarray)[1:10], each = 58),
    sex = patient$sex
    );

# Minimal input
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Minimal_Input', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Minimal input',
    xaxis.rot = 90,
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Axes & Labels
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Axes_Labels', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Axes & labels',
    xaxis.rot = 90,
    # Adjusting axes
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 13),
```

```
    yat = seq(0, 12, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Range
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Range', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Range',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    # adjusted y-axis limits
    ylimits = c(0, 11),
    yat = seq(0, 10, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # Specify range
    start = 1,
    end = 10,
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Scaling
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Scale', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Scale',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 13),
    yat = seq(0, 12, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
```

```
    # Scale
    scale = TRUE,
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );


# Extra points
median.points <- unlist(tapply(violin.data$values, violin.data$genes, median));
top.points <- unlist(tapply(violin.data$values, violin.data$genes, quantile, 0.90));

create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Points', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Extra points',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 13),
    yat = seq(0, 12, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    # Adding median and 90th percentile
    extra.points = list(median.points, top.points),
    extra.points.pch = 21,
    extra.points.col = c('white','grey'),
    extra.points.cex = 0.5,
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Colours
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Colour', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Colour',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 13),
    yat = seq(0, 12, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
```

```
    extra.points = list(median.points, top.points),
    extra.points.pch = 21,
    extra.points.col = c('white','grey'),
    extra.points.cex = 0.5,
    # Colour
    col = 'dodgerblue',
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Custom labels
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Custom_Labels', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Custom labels',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 16),
    yat = c(0,1,2,4,8,16),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.label = 'Gene',
    ylab.label = 'Change in expression',
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    extra.points = list(median.points, top.points),
    extra.points.pch = 21,
    extra.points.col = c('white','grey'),
    extra.points.cex = 0.5,
    col = 'dodgerblue',
    # customizing labels
    yaxis.lab = c(
        0,
        expression(paste('2'^'0')),
        expression(paste('2'^'1')),
        expression(paste('2'^'2')),
        expression(paste('2'^'4')),
        expression(paste('2'^'5'))
        ),
    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 100
    );

# Orientation
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Orientation', fileext = '.tiff'),
    # switch formula
    formula = genes ~ values,
    data = violin.data,
    main = 'Orientation',
    xaxis.rot = 90,
```

```
        xaxis.cex = 1,
        yaxis.cex = 1,
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        ylab.label = 'Gene',
        xlab.label = 'Change in expression',
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        extra.points = list(median.points, top.points),
        extra.points.pch = 21,
        extra.points.col = c('white','grey'),
        extra.points.cex = 0.5,
        col = 'dodgerblue',
        # orientation
        plot.horizontal = TRUE,
        description = 'Violinplot created by BoutrosLab.plotting.general',
        resolution = 100
        );

    # background
    create.violinplot(
        # filename = tempfile(pattern = 'Violinplot_Background', fileext = '.tiff'),
        formula = values ~ genes,
        data = violin.data,
        main = 'Background rectangle',
        xaxis.rot = 90,
        xaxis.cex = 1,
        yaxis.cex = 1,
        ylimits = c(0, 13),
        yat = seq(0, 12, 2),
        xaxis.fontface = 1,
        yaxis.fontface = 1,
        xlab.label = 'Gene',
        ylab.label = 'Change in expression',
        xlab.cex = 1.5,
        ylab.cex = 1.5,
        extra.points = list(median.points, top.points),
        extra.points.pch = 21,
        extra.points.col = c('white','grey'),
        extra.points.cex = 0.5,
        col = 'dodgerblue',
        # background
        add.rectangle = TRUE,
        xleft.rectangle = seq(0.5, 8.5, 2),
        ybottom.rectangle = 0,
        xright.rectangle = seq(1.5, 9.5, 2),
        ytop.rectangle = 13,
        col.rectangle = 'grey',
        alpha.rectangle = 0.5,
        description = 'Violinplot created by BoutrosLab.plotting.general',
        resolution = 100
        );
```

```
# Nature style
create.violinplot(
    # filename = tempfile(pattern = 'Violinplot_Nature_style', fileext = '.tiff'),
    formula = values ~ genes,
    data = violin.data,
    main = 'Nature style',
    xaxis.rot = 90,
    xaxis.cex = 1,
    yaxis.cex = 1,
    ylimits = c(0, 13),
    yat = seq(0, 12, 2),
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    col = 'dodgerblue',
    add.rectangle = TRUE,
    xleft.rectangle = seq(0.5, 8.5, 2),
    ybottom.rectangle = 0,
    xright.rectangle = seq(1.5, 9.5, 2),
    ytop.rectangle = 13,
    col.rectangle = 'grey',
    alpha.rectangle = 0.5,

    # set style to Nature
    style = 'Nature',

    # demonstrating how to italicize character variables
    ylab.lab = expression(paste('italicized ', italic('a'))),

    # demonstrating how to create en-dashes
    xlab.lab = expression(paste('en dashs: 1','\u2013', '10'^'\u2013', ''^3)),

    description = 'Violinplot created by BoutrosLab.plotting.general',
    resolution = 200
    );
```

`critical.value.ks.test` *Critical Value for Kolmogorov-Smirnov Test*

Description

Takes a sample size and a confidence level and computes the corresponding critical value basing on the kolmogorov-smirnov test

Usage

```
critical.value.ks.test(n, conf, alternative = "two.sided");
```

Arguments

`n`	The sample size
`conf`	The confidence level
`alternative`	Indicates the alternative hypothesis and must be one of "two.sided"(default), "one-sided".

Value

The corresponding critical value

Author(s)

Ying Wu

Examples

```
critical.value.ks.test(10, 0.95);
critical.value.ks.test(100, 0.95, alternative = "one-sided");
```

`default.colours` *Provides default colour schemes.*

Description

Returns colour schemes based on user input. Used to provide default colour schemes for simple cases.

Usage

```
default.colours(
number.of.colours = 2,
palette.type = 'qual',
is.greyscale = TRUE,
is.venn = FALSE
);
```

Arguments

`number.of.colours`	The number of colours requested for the colour scheme.
`palette.type`	The type of colour scheme requested. Only palette types of “seq”, “div”, “qual”, “pastel”, “survival”, “dotmap”, “spiral.sunrise”, “spiral.morning”, “spiral.dusk”, “spiral.noon”, “spiral.afternoon”, “spiral.dawn”, and “spiral.night” are accepted. Legacy colour palettes are available under “chromosomes”, “old.qual1”, “old.qual2”, “old.seq”, and “old.div”. “seq” corresponds to sequential colour schemes, “div” corresponds to diverging colour schemes, and “qual” corresponds to qualitative colour schemes - “pastel” is a pastel version of this palette. “survival” is useful

	for survival plots, as the first two colour are blue and red, following convention. The remaining colour schemes are not tied to a specific use-case.
`is.greyscale`	Boolean asking whether or the colour scheme should be greyscale-compatible. Defaults to TRUE. The purpose of this parameter is to warn users if they ask for a colour scheme that is not greyscale-compatible. Regardless of the value of is.greyscale, the same colour scheme will be provided.
`is.venn`	Boolean determining whether or not the colour scheme is to be used for a venn diagram. If TRUE, the palette type should be set to NULL. For venn diagrams, text colours are also provided.

Details

For further information on colour schemes, refer to the plotting guide.)

Author(s)

Christine P’ng

Examples

```
default.colours(number.of.colours = 6, is.greyscale = FALSE, palette.type = 'div')
# Returns:
# [1] ”#B32B2B” ”#DD4E4E” ”#EB7C7C” ”#F7BEBE” ”#BEF4F7” ”#80CDD1”

default.colours(number.of.colours = 3, palette.type = NULL, is.venn = TRUE)
# Returns:
# [1] ”red”        ”dodgerblue” ”yellow”
# [1] ”darkred”    ”darkblue”   ”darkorange”
# The second line of colours is the corresponding text colour

default.colours(number.of.colours = c('2','5','3'), c('binary','seq','seq'))
# Returns:
# [[1]]
# [1] ”white”       ”chartreuse3”

# [[2]]
# [1] ”lavenderblush”  ”pink”           ”palevioletred1” ”violetred1”
# [5] ”maroon”

# [[3]]
# [1] ”aliceblue”    ”lightblue1”   ”lightskyblue”

default.colours(5, 'spiral.sunrise');
# Returns:
# [1] ”#336A90” ”#65B4A2” ”#B1D39A” ”#F4E0A6” ”#FFE1EE”
```

`display.colours` *Function to display R colors, as well as corresponding R grey colours.*

Description

Displays R colors and their corresponding R grey colours.

Usage

```
display.colours(
cols,
names = cols
);
```

Arguments

`cols`	Vector of colours to be displayed.
`names`	The names of the colours. Defaults to equal the input of cols

Details

For further information on colour schemes, refer to the colour guide (in Resources/general)

Author(s)

Christine P'ng

Examples

```
display.colours('red');
# Red and Grey are displayed

display.colours(default.colours(5));
# Five default colours and their grey values are displayed

test.colours <- force.colour.scheme(c('skin','nerve'), 'tissue');
display.colours(test.colours);
```

`display.statistical.result` *Utility function to display statistical result in a plot*

Description

A utility function to display statistical result in a plot in scientific notation (when appropriate)

Usage

```
display.statistical.result(
x,
lower.cutoff = 2.2e-50,
scientific.cutoff = 0.001,
digits = 2,
statistic.type = 'P',
symbol = ': '
);
```

Arguments

`x`	Numeric value to be displayed
`lower.cutoff`	For values of x smaller than lower.cutoff, the return value will be "< lower.cutoff". Defaults to 2.2e-16
`scientific.cutoff`	For values of x larger or equal to scientific.cutoff, standard notation will be used (rather than scientific notation). Defaults to 0.001
`digits`	Number of decimal places of precision to be shown
`statistic.type`	Type of statistic to be displayed, defaults to “P”.
`symbol`	Symbol prior to statistic to be displayed, defaults to “: ”.

Value

Returns an expression

Author(s)

Nathalie Moon

See Also

`scientific.notation`

Examples

```
set.seed(100);

display.statistical.result(x = 0.0000000000000000000234);
display.statistical.result(x = 0.023, statistic.type = 'Q');
display.statistical.result(x = 0.001, scientific.cutoff = 0.01, symbol = ' = ');
```

`dist` *Distance Matrix Computation*

Description

This function computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.

Usage

```
dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)
```

Arguments

`x`	a numeric matrix, data frame or `"dist"` object.
`method`	the distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"`, `"minkowski"`, or `"jaccard"`. Any unambiguous substring can be given.
`diag`	logical value indicating whether the diagonal of the distance matrix should be printed by `print.dist`.
`upper`	logical value indicating whether the upper triangle of the distance matrix should be printed by `print.dist`.
`p`	The power of the Minkowski distance.

Details

Available distance measures are (written for two vectors x and y):

`euclidean`**:** Usual square distance between the two vectors (2 norm).

`maximum`**:** Maximum distance between two components of x and y (supremum norm)

`manhattan`**:** Absolute distance between the two vectors (1 norm).

`canberra`**:** $\sum_i |x_i - y_i|/|x_i + y_i|$. Terms with zero numerator and denominator are omitted from the sum and treated as if the values were missing.

This is intended for non-negative values (e.g. counts): taking the absolute value of the denominator is a 1998 R modification to avoid negative distances.

`binary`**:** (aka *asymmetric binary*): The vectors are regarded as binary bits, so non-zero elements are ‘on’ and zero elements are ‘off’. The distance is the *proportion* of bits in which only one is on amongst those in which at least one is on.

`minkowski`**:** The p norm, the pth root of the sum of the pth powers of the differences of the components.

`jaccard`**:** The proportion of items that are not in both sets. For binary data, the output is equal to dist(method ="binary")

Missing values are allowed, and are excluded from all computations involving the rows within which they occur. Further, when `Inf` values are involved, all pairs of values are excluded when their contribution to the distance gave `NaN` or `NA`. If some columns are excluded in calculating a Euclidean, Manhattan, Canberra or Minkowski distance, the sum is scaled up proportionally to the number of columns used. If all pairs are excluded when calculating a particular distance, the value is `NA`.

The `"dist"` method of `as.matrix()` and `as.dist()` can be used for conversion between objects of class `"dist"` and conventional distance matrices.

Value

`dist` returns an object of class `"dist"`.

The lower triangle of the distance matrix stored by columns in a vector, say `do`. If `n` is the number of observations, i.e., `n <- attr(do, "Size")`, then for $i < j \le n$, the dissimilarity between (row) i and j is `do[n*(i-1) - i*(i-1)/2 + j-i]`. The length of the vector is $n * (n - 1)/2$, i.e., of order n^2.

The object has the following attributes (besides `"class"` equal to `"dist"`):

`Size`	integer, the number of observations in the dataset.
`Labels`	optionally, contains the labels, if any, of the observations of the dataset.
`Diag`, `Upper`	logicals corresponding to the arguments `diag` and `upper` above, specifying how the object should be printed.
`call`	optionally, the `call` used to create the object.
`method`	optionally, the distance method used; resulting from `dist()`, the (`match.arg()`ed) `method` argument.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979) *Multivariate Analysis.* Academic Press.

Borg, I. and Groenen, P. (1997) *Modern Multidimensional Scaling. Theory and Applications.* Springer.

See Also

`daisy` in the cluster package with more possibilities in the case of *mixed* (continuous / categorical) variables. `hclust`.

Examples

```
x <- matrix(rnorm(100), nrow=5)
dist(x)
dist(x, diag = TRUE)
dist(x, upper = TRUE)
m <- as.matrix(dist(x))
d <- as.dist(m)
stopifnot(d == dist(x))

## Use correlations between variables "as distance"
dd <- as.dist((1 - cor(USJudgeRatings))/2)
round(1000 * dd) # (prints more nicely)
plot(hclust(dd)) # to see a dendrogram of clustered variables

## example of binary and canberra distances.
x <- c(0, 0, 1, 1, 1, 1)
y <- c(1, 0, 1, 1, 0, 1)
dist(rbind(x,y), method= "binary")
## answer 0.4 = 2/5
dist(rbind(x,y), method= "canberra")
## answer 2 * (6/5)
dist(rbind(x,y), method= "jaccard")
## answer 0.4 = 2/5

## To find the names
labels(eurodist)

## Examples involving "Inf" :
## 1)
x[6] <- Inf
(m2 <- rbind(x,y))
dist(m2, method="binary")# warning, answer 0.5 = 2/4
## These all give "Inf":
stopifnot(Inf == dist(m2, method= "euclidean"),
          Inf == dist(m2, method= "maximum"),
          Inf == dist(m2, method= "manhattan"))
##  "Inf" is same as very large number:
x1 <- x; x1[6] <- 1e100
stopifnot(dist(cbind(x ,y), method="canberra") ==
    print(dist(cbind(x1,y), method="canberra")))

## 2)
y[6] <- Inf #-> 6-th pair is excluded
dist(rbind(x,y), method="binary")   # warning; 0.5
dist(rbind(x,y), method="canberra") # 3
dist(rbind(x,y), method="maximum")  # 1
dist(rbind(x,y), method="manhattan")# 2.4
```

`force.colour.scheme` *Based on predefined colour schemes, returns a vector of corresponding colours.*

Description

Takes a vector of character strings and an scheme returns the matching colours as a vector.

Usage

```
force.colour.scheme(
x = NA,
scheme,
fill.colour = 'slategrey',
include.names = FALSE,
return.factor = FALSE,
return.scheme = FALSE
);
```

Arguments

`x`	The input character or numeric vector, defaults to NA incase return.scheme = TRUE.
`scheme`	A string representing a predefined scheme. Available schemes are: “annovar.annotation”, “annovar.annotation.collapsed”, “annovar.annotation.collapsed2”, “tissue”, “sex”, “stage”, “risk”, “MSI”, “tumour”, “CNV”, “organism”

, “chromosome” and “biomolecule”

`fill.colour`	Value to enter when value of x not present in scheme.
`include.names`	Should the output be a named vector or not?
`return.factor`	Should factors (scheme names) be returned?
`return.scheme`	Should the scheme list be returned?

Details

The input character options for each colour scheme are as follows: annovar.annotation

- nonsynonymous snv
- stopgain snv
- stoploss snv
- frameshift deletion
- frameshift substitution
- splicing
- synonymous snv

annovar.annotation.collapsed

- nonsynonymous snv
- stopgain snv
- stoploss SNV
- frameshift indel

- splicing

annovar.annotation.collapsed2

- nonsynonymous
- stopgain-stoploss
- splicing
- frameshift indel
- synonymous
- utr5-utr3
- nonframeshift indel
- intronic
- intergenic
- other

tissue

- cartilage
- bone
- adipose
- bladder
- kidney
- blood
- heart
- muscle
- hypothalamus
- pituitary
- thyroid
- parathyroid
- skin
- salivarygland
- esophagus
- stomach
- liver
- gallbladder
- pancreas
- intestine
- colon
- pharynx
- larynx

- trachea
- diaphragm
- lung
- nerve
- spine
- brain
- eye
- breast
- ovary
- uterus
- prostate
- testes
- lymph
- leukocyte
- spleen

sex

- male
- female

stage

- I
- II
- III
- IV

risk

- High
- Low

MSI

- MSI-High
- MSI-Low
- MSS

tumour

- Primary
- Metastatic

CNV

- Amplification
- Deletion
- LOH
- Neutral

organism

- Human
- Rat
- Mouse

chromosome

- 1 - 22
- X
- Y

biomolecule

- DNA
- RNA
- Protein
- Carbohydrate
- Lipid

clinicalt3

- t0
- t1
- t2
- t3
- t4
- t5

clinicalt9

- t1a
- t1b
- t1c
- t2a
- t2b
- t2c
- t3a
- t3b
- t3c

gleason.score

- 3+3
- 3+4
- 4+4
- 4+5
- 3+5
- 5+3
- 5+4
- 5+5
- missing
- NA

gleason.sum

- 5
- 6
- 7
- 8
- 9
- missing
- NA

tissue.color

- blood
- frozen
- ffpe

psa.categorical

- 0 - 9.9
- 10 - 19.9
- >= 20

age.categorical.default

- <50
- 50 - 60
- 60 - 70
- >= 70

age.categorical.prostate

- <40

- 40 - 50
- 50 - 65
- 65 - 70
- >= 70

age.gradient

psa.gradient

heteroplasmy

- 0 - 0.2
- 0.2 - 0.4
- 0.4 - 0.6
- 0.6 - 1.0

mt.annotation

- MT-DLOOP
- MT-T*
- MT-RNR*
- MT-ND1
- MT-ND2
- MT-ND3
- MT-ND4L
- MT-ND4L/MT-ND4
- MT-ND4
- MT-ND5
- MT-ND6
- MT-CO1
- MT-CO2
- MT-CO3
- MT-ATP6/MT-CO3
- MT-ATP6
- MT-ATP8/MT-ATP6
- MT-ATP8
- MT-CYB
- MT-NC*
- MT-OL*

isup.grade

- 1
- 2
- 3
- 4
- 5

Value

If multiple returns are requested, outputs a list (return.factor: factor length x with scheme names; scheme: list containing scheme names and colours; colours: vector length x with the required colours).

Author(s)

Nicholas Harding

Examples

```
annovar.output <- c('nonsynonymous snv', 'stopgain snv', 'none', 'stoploss snv',
'frameshift deletion', 'frameshift substitution', 'splicing', 'none');
force.colour.scheme(annovar.output,'annovar.annotation');
force.colour.scheme(annovar.output,'annovar.annotation', 'white');
```

`generate.at.final` *Generates alternative default tick mark locations for create.densityplot() and create.scatterplot()*

Description

Generates the tick mark locations for the output graphic of create.densityplot(), based on the values to the arguments of that function. This is needed to ensure the grid line and tick mark locations agree with each other.

Usage

```
generate.at.final(
at.input,
limits,
data.vector
);
```

Arguments

`at.input`	either a logical scalar or a numeric vector
`limits`	either NULL or a numeric vector of length 2
`data.vector`	a numeric vector

Value

Returns a numeric vector containing the tick mark locations of the densityplot.

Author(s)

Kenneth C.K. Chu

`get.corr.key` *Correlation Key*

Description

A function for adding correlation key legends to scatterplots.

Usage

```
get.corr.key(
x,
y,
label.items = c(”spearman”, ”spearman.p”),
x.pos = 0.03,
y.pos = 0.97,
key.corner = NULL,
key.cex = 1,
key.title = NULL,
title.cex = 1,
alpha.background = 0,
num.decimals = 2,
        border = 'white'
)
```

Arguments

`x`	A vector of values
`y`	Another vector of values with the same length as x
`label.items`	A vector of things to include in the key. Any combination of the following can be used. c("spearman","pearson","kendall","beta0", beta1,"spearman.p","pearson.p","kendall.p","beta1.p", "beta.robust", "beta.robust.p"). "all" is an alternative to the full list.
`x.pos`	Horizontal position of the key corner
`y.pos`	Vertical position of the key corner
`key.corner`	The corner of the key defaults to the closest corner of the plot. This helps overcome some variable character and row sizing.
`key.cex`	Specifies the size of font for the key, defaults to 1.
`key.title`	The title of the key. Defaults to NULL
`title.cex`	The size of the key title. Defaults to 1
`alpha.background`	A value from 0 to 1 indicating the transparency of the legend box.
`num.decimals`	Number of decimal places to keep for spearman, pearson and kendall correlations. Defaults to 2.
`border`	Adds border around the key with the color specified, alpha background cannot be 0. Defaults to White.

Value

Returns a key in the format specified in the xyplot documentation.

Author(s)

Daryl Waggott

See Also

xyplot, plotmath

Examples

```
# create some temporary data
tmp.data <- data.frame(
    x = c(
        runif(n = 15, min =  0, max =  20),
        runif(n = 15, min = 80, max = 100),
        runif(n = 70, min =  0, max = 100)
        ),
    y = c(
        runif(n = 15, min =  0, max =  20),
        runif(n = 15, min = 80, max = 100),
        runif(n = 70, min =  0, max = 100)
        )
    );

# a simple scatterplot with correlation key
BoutrosLab.plotting.general::create.scatterplot(
    formula = y ~ x,
    data = tmp.data,
    # filename = tempfile(pattern = 'get.corr.key-scatterplot', fileext = '.tiff'),
    xlab.label = 'X Axis Title',
    ylab.label = 'Y Axis Title',
    xlimits = c(0,100),
    ylimits = c(0,100),
    xat = seq(0,100,25),
    yat = seq(0,100,25),
    add.axes = FALSE,
    key = BoutrosLab.plotting.general::get.corr.key(
        tmp.data$y,
        tmp.data$x,
        label.items = c('spearman', 'spearman.p', 'kendall', 'beta1', 'beta1.p')
        )
    );


# compare beta1 vs a robust estimate of the slope

# add an outlier
tmp.data <- rbind(tmp.data, c(2000,100));
```

```
BoutrosLab.plotting.general::create.scatterplot(
    formula = y ~ x,
    data = tmp.data,
   # filename = tempfile(pattern = 'get.corr.key.robust-scatterplot', fileext = '.tiff'),
    xlab.label = 'X Axis Title',
    ylab.label = 'Y Axis Title',
    xlimits = c(0,100),
    ylimits = c(0,100),
    xat = seq(0,100,25),
    yat = seq(0,100,25),
    add.axes = FALSE,
    key = BoutrosLab.plotting.general::get.corr.key(
        tmp.data$y,
        tmp.data$x,
        label.items = c('beta1', 'beta1.robust','beta1.p','beta1.robust.p')
        )
    );

# see create.scatterplot for an example of creating multiple keys using legends
```

`get.correlation.p.and.corr` *Calculate a correlation and its statistical significance*

Description

Returns the correlation and p-value for two variables using a user-specified correlation metric. P-values are estimated analytically, not via permutation-testing.

Usage

```
get.correlation.p.and.corr(x, y, alternative = 'two.sided', method = 'pearson');
```

Arguments

`x`	Vector of numbers to analyze
`y`	Vector of numbers to analyze
`alternative`	What is the null-hypothesis?
`method`	The correlation technique to use (passed directly to cor.test)

Value

Returns a two-element vector containing the correlation and its p-value.

Author(s)

Paul C. Boutros

Examples

```
get.correlation.p.and.corr(
x = runif(100),
y = runif(100),
method = 'pearson'
);

get.correlation.p.and.corr(
x = sample(1:10, 100, replace = TRUE),
y = runif(100),
method = 'spearman'
);
```

`get.defaults` *Get operating system specific default properties*

Description

Returns the value for the property requested

Usage

```
get.defaults(
property = 'fontfamily',
os.type = .Platform$OS.type,
add.to.list = NULL,
use.legacy.settings = FALSE
);
```

Arguments

`property`	The property to be retrieved
`os.type`	operating system (optional). valid values are: “windows”, “unix”
`add.to.list`	appends the requested property to this parameter
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)

Value

Returns the value (list if add.to.list is passed) for the property requested given the os.type parameter. If latter is missing, it attempts to find user’s operating system

Author(s)

Syed Haider

Examples

```
# returns the fontfamily for current OS
get.defaults(property = "fontfamily");

# returns the fontfamily for unix
get.defaults(property = "fontfamily", os.type = 'windows');
```

`get.line.breaks` *Get line breaks*

Description

Given a vector, returns the indices (and an adjustment to draw lines between cells) where the value is not equal to the preceding value. Main use intended to be in row.lines arguments to create.heatmap

Usage

```
get.line.breaks(
x
);
```

Arguments

`x`	A vector, numeric, factor or character.

Value

A vector of integers representing the break points in the vector x

Author(s)

Nicholas Harding

Examples

```
set.seed(12345);
values <- sample(
default.colours(3),
20,
replace = TRUE
);
get.line.breaks(values);
```

`legend.grob` *Generate a legend grob*

Description

Takes a list and generates a grob representing one or more legends

Usage

```
legend.grob(
legends,
label.cex = 1,
title.cex = 1,
title.just = 'centre',
title.fontface = 'bold',
font.family = NULL,
size = 3,
border = NULL,
border.padding = 1,
layout = c(1, length(legends)),
between.col = 1,
between.row = 1,
use.legacy.settings = FALSE,
        x = 0.5,
        y = 0.5,
background.col = "white",
background.alpha = 0
);
```

Arguments

`legends`	A list defining one or more legends. Each must be a separate component called 'legend'. Each component is a list with components 'colours', 'labels', 'border' (optional), 'title' (optional), and 'size' (optional). The 'colours' component is a vector of fill colours to be used for the rectangles, the 'labels' component is a vector of text labels corresponding to the colours, the 'border' component specifies the colours of the rectangle borders (defaults to black), and the 'title' component is a character string representing a title for the legend.
`label.cex`	Size of text labels in the legends, defaults to 1.
`title.cex`	Size of titles in the legends, defaults to 1.
`title.just`	Justification of titles in the legends. Defaults to 'centre'.
`title.fontface`	Font face of titles in the legends ('plain', 'bold', 'italic', etc.)
`font.family`	Font to be used for legend text. If NULL, the default font is used.
`size`	Width of the legend boxes in 'character' units. If a 'size' argument is specified for a legend component, it will override this value.

`border`	A list of parameters (passed to `gpar`) specifying line options for the legend border. If NULL, no border is drawn.
`border.padding`	The amount of empty space (split equally on both sides) to add between the legend and its border, in 'lines' units. Defaults to 1.
`layout`	Numeric vector of length 2 specifying the number of columns and rows for the legend layout. Defaults to a 1-column layout. Note that legends are added to the layout in a row-wise order.
`between.col`	Amount of space to add between columns in the layout, in 'lines' units. Defaults to 0.5.
`between.row`	Amount of space to add between rows in the layout, in 'lines' units. Defaults to 0.5.
`use.legacy.settings`	boolean to set wheter or not to use legacy mode settings (font)
`x`	x coordinate in npc coordinate system
`y`	y coordinate in npc coordinate system
`background.col`	colour for the background of the legend grob
`background.alpha`	alpha for the background of the legend grob

Value

Returns an grob representing the legend(s)

Implementation

This function was initially created to be called from `create.heatmap` to draw a covariate legend.
The decision to use a grob (grid graphical object) to represent the legend was made based on the format of the `levelplot` function in the `lattice` package. Since the `legend` argument of the function requires grobs, it was easiest to create a grob to represent the legend and then, if necessary, add this to any existing grobs (dendrograms, etc.) in the `create.heatmap` function using a grid layout.
An alternative method of creating the legend using the `barchart` function was tested, but it was unclear how to merge this barchart with the heatmap since the `c.trellis` function attempts to unify the format of the two images, and the use of viewports required that the plots be drawn, eliminating the possibility of suppressing output and saving the final graph as a trellis object.

Author(s)

Lauren Chong

See Also

`create.heatmap`, `draw.key`, `gpar`

Examples

```
# The 'cairo' graphics is preferred but on M1 Macs this is not available
bitmap.type = getOption('bitmapType')
if (capabilities('cairo')) {
bitmap.type <- 'cairo';
}

# create list representing two legends
legends1 <- list(
    legend = list(
        colours = c('orange', 'chartreuse4', 'darkorchid4'),
        labels = c('Group 1', 'Group 2', 'Group 3'),
        border = c('orange', 'chartreuse4', 'darkorchid4'),
        title = 'Legend #1'
        ),
    legend = list(
        colours = c('firebrick3', 'lightgrey'),
        labels = c('Case', 'Control')
        )
    );

# create a legend grob using defaults
legend.grob1 <- legend.grob(
    legends = legends1
    );
tiff(
    filename = tempfile(pattern = 'legend_grob1', fileext = '.tiff'),
    type = bitmap.type,
    width = 5,
    height = 5,
    units = 'in',
    res = 800,
    compression = 'lzw'
    );
grid.draw(legend.grob1);
dev.off();

# create the same legend with some customizations
legend.grob2 <- legend.grob(
    legends = legends1,
    label.cex = 1.25,
    title.cex = 1.25,
    title.just = 'left',
    title.fontface = 'bold.italic',
    size = 4,
    border = list(),
    layout = c(2,1)
    );
tiff(
    filename = tempfile(pattern = 'legend_grob2', fileext = '.tiff'),
    type = bitmap.type,
    width = 5,
```

```
    height = 5,
    units = 'in',
    res = 800,
    compression = 'lzw'
    );
grid.draw(legend.grob2);
dev.off();

# create a legend where the title is underlined (see ?plotmath), add space between rows
legends2 <- list(
    legend = list(
        colours = c('orange', 'chartreuse4', 'darkorchid4'),
        labels = c('Group 1', 'Group 2', 'Group 3'),
        title = expression(underline('Legend #1'))
        ),
    # Use dots instead of rectangles
    point = list(
        colours = c('firebrick3', 'lightgrey'),
        labels = c('A label', 'A longer label'),
        # Set dot size
        cex = 1.5
        )
    );

# create the new legend and use more complex border
legend.grob3 <- legend.grob(
    legends = legends2,
    border = list(col = 'blue', lwd = 2, lty = 3),
    border.padding = 1.5,
    between.row = 3
    );
tiff(
    filename = tempfile(pattern = 'legend_grob3', fileext = '.tiff'),
    type = bitmap.type,
    width = 5,
    height = 5,
    units = 'in',
    res = 800,
    compression = 'lzw'
    );
grid.draw(legend.grob3);
dev.off();

# Make a legend where the size of boxes is customized
legends3 <- list(
    legend = list(
        colours = c('orange', 'chartreuse4', 'darkorchid4'),
        labels = c('Group 1', 'Group 2', 'Group 3'),
        title = 'Legend #1',
        size = c(3,2,1)
        ),
    legend = list(
        colours = NULL,
```

```
            labels = c('+', '-'),
            border = 'transparent',
            title = 'Disease status',
            size = 0.5
            )
        );
    legend.grob4 <- legend.grob(
        legends = legends3
        );
    tiff(
        filename = tempfile(pattern = 'legend_grob4', fileext = '.tiff'),
        type = bitmap.type,
        width = 5,
        height = 5,
        units = 'in',
        res = 800,
        compression = 'lzw'
        );
    grid.draw(legend.grob4);
    dev.off();
```

`microarray` *Microarray dataset of colon cancer patients*

Description

Gene expression level changes of 2382 genes across 58 colon cancer patients. Additional data on the genes include chromosomal location and p-values. Additional data on the patient samples is found in in the "`patient`" dataset. The same patient samples are described in the "`SNV`" and "`CNA`" datasets.

Usage

```
microarray
```

Format

A data frame with 62 columns and 2383 rows. Columns 1-58 indicate the cancer patient sample. Columns 59-61 indicate the (sorted) chromosomal location by "`Chr`", "`Start`", and "`End`". Column 62 contains adjusted p-values. Each row is a different gene, and the row names are the gene names.

Author(s)

Christine P’ng

Examples

```
create.dotmap(
    # filename = tempfile(pattern = 'Using_microarray_dataset', fileext = '.tiff'),
    x = microarray[1:5,1:5],
    main = 'microarray data',
    spot.size.function = function(x) {abs(x)/3;},
    xaxis.cex = 0.8,
        yaxis.cex = 0.8,
        xaxis.rot = 90,
    description = 'Dotmap created by BoutrosLab.plotting.general'
    );
```

`panel.BL.bwplot` *A lattice::panel.bwplot replacement that fixes colouring issues*

Description

Function lattice::bwplot() shows unexpected and unintuitive behaviour when colouring parameters of par.settings are vectors. The function panel.BL.bwplot fixes these issues. It should be called only from lattice::bwplot(). Use with caution. This function is invoked by create.boxplot

Arguments

`...`	Pass through argument. See lattice::bwplot() for further details.
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE

Author(s)

Mehrdad Shamsi

See Also

`create.boxplot`

`patient` *Dataset describing qualities of 58 colon cancer patients*

Description

A number of qualities describing 58 colon cancer patients. The same patient samples are described in the "`microarray`", "`SNV`" and "`CNA`" datasets.

Usage

```
patient
```

Format

A data frame with 5 columns and 58 rows. Each row indicates a different patient sample, with the following columns describing a feature of the sample:

sex The sex of the paient, either "male" or "female"

stage The stage of the patient's cancer, one of "I", "II", "III", "IV", or NA

msi The microsatellite instabiltiy of the cancer, either "MSS" or "MSI-High"

prop.CAGT The proportion of C to A or G to T base changes between the sample and reference genome

prop.CTGA The proportion of C to T or G to A base changes between the sample and reference genome

prop.CGGC The proportion of C to G or G to C base changes between the sample and reference genome

prop.TAAT The proportion of T to A or A to T base changes between the sample and reference genome

prop.TGAC The proportion of T to G or A to C base changes between the sample and reference genome

prop.TCAG The proportion of T to C or A to G base changes between the sample and reference genome

Author(s)

Christine P'ng

Examples

```
# use sample to set colour scheme
sex.colours <- replace(as.vector(patient$sex), which(patient$sex == 'male'),'dodgerblue');
sex.colours <- replace(sex.colours, which(patient$sex == 'female'), 'pink');
len <- apply(SNV[1:15], 2, function(x){mutation.count <- length(which(x == 1))});

create.barplot(
    # filename = tempfile(pattern = 'Using_patient_dataset', fileext = '.tiff'),
    formula = len ~ colnames(SNV[1:15]) ,
    data = SNV,
    main = 'patient dataset',
    xaxis.rot = 45,
    ylimits = c(0,30),
    yat = seq(0,30,5),
    col = sex.colours,
    description = 'Barplot created by BoutrosLab.plotting.general'
    );
```

`pcawg.colours` *Return standard PCAWG colour palettes.*

Description

Return standard PCAWG colour palettes. Case insensitive.

Usage

```
pcawg.colours(
x = NULL,
scheme = NULL,
fill.colour = 'slategrey',
return.scheme = FALSE);
```

Arguments

`x`	Chracter vector with terms to be mapped to colours. Ignored if scheme='all' or return.scheme=TRUE.
`scheme`	String specifying desired colour scheme. To see all available schemes, use scheme='all', returns.scheme=FALSE.
`fill.colour`	Unrecognized output will be filled with this colour. Default to 'slategrey'.
`return.scheme`	TRUE/FALSE. Set to true to return full specified scheme. Set to false to map x to colours.

Details

For further information on colour schemes, refer to the plotting guide.)

Author(s)

Jennifer Aguiar & Constance Li

`scientific.notation` *Use scientific notation in plots*

Description

Returns an expression or list for plotting data in scientific notation

Usage

```
scientific.notation(
x,
digits = 1,
type = 'expression'
);
```

Arguments

`x`	The number we want in scientific notation.
`digits`	How many decimal places to keep.
`type`	The format to return the value in. Defaults to ’expression’, also accepts ’list’

Value

Generates scientific notation either as an expression or list.

Author(s)

Paul C. Boutros

---

show.available.palettes
Display the available colour palettes

Description

Displays the available colour palettes

Usage

```
show.available.palettes(
type = 'general',
filename = NULL,
height = 5,
width = 8,
resolution = 300
);
```

Arguments

`type`	Either “general”, “specific”, or “both” (default)
`filename`	Filename for tiff output, or if NULL returns the trellis object itself
`height`	Figure height, defaults to 8 inches – this is optimal for the specific schemes
`width`	Figure width, defaults to 12 – this is optimal for the specific schemes
`resolution`	Figure resolution in dpi, defaults to 300

Author(s)

Christine P’ng

Examples

```
show.available.palettes(
# filename = tempfile(pattern = 'show_case_specific_schemes', fileext = '.tiff'),
type = 'specific',
width = 10
);

show.available.palettes(
# filename = tempfile(pattern = 'default_schemes', fileext = '.tiff'),
type = 'general',
height = 6,
width = 8
);
```

`SNV` *Single nucleotide variant (SNV) data from colon cancer patients*

Description

SNV calls from 30 genes across 58 colon cancer patients. Additional data on the patient samples is found in in the "`patient`" dataset. The same patient samples are described in the "`microarray`" and "`CNA`" datasets.

Usage

```
SNV
```

Format

A data frame with 58 columns and 30 rows. The columns indicate the patient sample, and the rows indicate the gene. The contents of the data frame are either NA (indicating no SNV call was made) or one of:

- 1 - nonsynonymous SNV
- 2 - stopgain SNV
- 3 - frameshift insertion
- 4 - frameshift deletion
- 5 - nonframeshift insertion
- 6 - nonframeshift deletion
- 7 - splicing
- 8 - unknown

Author(s)

Christine P’ng

Examples

```
len <- apply(SNV[1:15], 2, function(x){mutation.count <- length(which(x == 1))});

create.barplot(
    # filename = tempfile(pattern = 'Using_SNV_dataset', fileext = '.tiff'),
    formula = len ~ colnames(SNV[1:15]) ,
    data = SNV,
    main = 'SNV dataset',
    xaxis.rot = 45,
    ylimits = c(0,30),
    yat = seq(0,30,5),
    description = 'Barplot created by BoutrosLab.plotting.general'
    );
```

`thousands.split` *Divide strings into groups of thousands*

Description

Takes a single number or list, and converts them into a new string with commas to mark the thousand multiples

Usage

```
thousands.split(
        nums
)
```

Arguments

`nums`	The numbers to be divided

Author(s)

Jeffrey Green

Examples

```
thousands.split(2344)

nums = c(1,2,34343,56565645645,676756,3434)

thousands.split(nums)

scatter.data <- data.frame(
    sample.one = microarray[1:800,1],
    sample.two = microarray[1:800,2],
    chr = microarray$Chr[1:800]
    );
```

```
create.scatterplot(
    # filename = tempfile(pattern = 'Test_Divide_Thousands', fileext = '.tiff'),
    formula = sample.two ~ sample.one,
    data = scatter.data,
    main = 'Axes & Labels',
    # Axes and labels
    xlab.label = colnames(microarray[1]),
    ylab.label = colnames(microarray[2]),
    yaxis.lab = thousands.split(c(1,2323,4545,567676,454,76767678678,89,787)),
    xat = seq(0, 16, 2),
    yat = seq(0, 16, 2),
    xlimits = c(0, 15),
    ylimits = c(0, 15),
    xaxis.cex = 1,
    yaxis.cex = 1,
    xaxis.fontface = 1,
    yaxis.fontface = 1,
    xlab.cex = 1.5,
    ylab.cex = 1.5,
    description = 'Scatter plot created by BoutrosLab.plotting.general'
    );
```

`write.metadata` *Writes Metadata*

Description

Utilizes exiftool to write metadata to generated plots. Writes the R version, lattice version, latticeExtra version, BoutrosLab.plotting.general version, BoutrosLab.plotting.survival version, operating system, machine, author, image description.

Usage

```
write.metadata(
filename = NULL,
description = NULL,
verbose = FALSE
);
```

Arguments

`filename`	Filename for output, or if NULL (default value) returns image unchanged.
`description`	Short description of image; default NULL
`verbose`	Option to standard output; default FALSE

Value

If `filename` is `NULL`, returns the image unchanged. If `description` is `NULL`, then the image is returned without the description tag.

Note: an easy way to view the metadata is by using the exiftool command.

Author(s)

Esther Jung

`write.plot`	*Simplifies plotting by standardizing and centralizing all output-handling*

Description

Handle various graphics-driver weirdness and writes an output file and returns 1 or returns the trellis.object

Usage

```
write.plot(
trellis.object,
filename = NULL,
additional.trellis.objects = NULL,
additional.trellis.locations = NULL,
height = 6,
width = 6,
size.units = 'in',
resolution = 1000,
enable.warnings = FALSE,
description = "Created with BoutrosLab.plotting.general"
);
```

Arguments

`trellis.object`	A trellis object to be plotted
`filename`	Filename for output, or if NULL (default value) returns the trellis object itself. Will automatically grab the extension used.
`additional.trellis.objects`	List of additional trellis objects to add to main plot. Default to NULL
`additional.trellis.locations`	List of coordinates for additional trellis objects. Must be represented using variable names ’xleft’, ’ybottom’, ’xright’ and ’ytop’. Defaults to NULL
`height`	Figure height, defaults to 6 inches
`width`	Figure width, defaults to 6 inches

`size.units`	Figure units, defaults to 'in'
`resolution`	Figure resolution, defaults to 1000
`enable.warnings`	Print warnings if set to TRUE, defaults to FALSE
`description`	Short description of image; default NULL

Value

Returns the trellis.object if filename is NULL or writes the plot to file if a filename is specified.

Author(s)

Paul C. Boutros

Examples

```
set.seed(253647)
# create test data
tmp.data <- data.frame(
    x = c(
        runif(n = 150, min =  0, max = 20),
        runif(n = 150, min = 40, max = 60),
        runif(n = 700, min =  0, max = 40)
        ),
    y = c(
        runif(n = 150, min =  0, max =  20),
        runif(n = 150, min = 40, max = 60),
        runif(n = 700, min =  0, max = 40)
        )
    );

main.plot <- create.densityplot(
    x = list(
        X = tmp.data$x,
        Y = tmp.data$y
        ),
    xlab.label = 'X Axis Title',
    ylab.label = 'Y Axis Title',
    xlimits = c(-50,150),
    ylimits = c(0,0.03),
    xat = seq(-50,150,50),
    yat = seq(0,0.03,0.005),
    description = 'Image description goes here'
    );

secondary.plot <- create.densityplot(
    x = list(
        X = tmp.data$x,
        Y = tmp.data$y
        ),
    xlab.label = '',
```

```
        ylab.label = '',
        xlimits = c(50,75),
        ylimits = c(0,0.015),
        xat = seq(0,150,10),
        yat = seq(0,0.015,0.005),
        xaxis.tck = 0,
        description = 'Image description goes here'
        );

    write.plot(
        filename = tempfile(pattern = 'write_plot_example', fileext = '.tiff'),
        trellis.object = main.plot,
        additional.trellis.objects = list(secondary.plot),
        additional.trellis.locations = list(
            xleft = 0.6,
            ybottom = 0.5,
            xright =0.97,
            ytop = 0.9
            ),
        resolution = 50 # Lowering resolution decreases file size
        );
```

Index