Package 'Gmisc'

July 28, 2014

Version 0.6.7

Date 2014-07-27

Title Collection of functions for plotting relations, generating tables, and more.

Author Max Gordon <max@gforge.se>

Maintainer Max Gordon <max@gforge.se>

Description This is a collection of functions for tables, plots, and more. For tables you'll find the convenient htmlTable() that I use for advanced markdown table layout. A major focus has been to have it compatible with LibreOffice and MS Word - you can now copy-paste most tables directly into your document from the viewer. The main plots in the package are the transitionPlot() and forestplot2(), see the examples for demo. Apart from those it is worth mentioning the getDescriptionStatsBy() and its associated function that help you generate a descriptive table (i.e. the Table 1 in most journals), the pvalueFormatter() that formats p-values, and mergeLists() that provides a complex recursive list merge.

License GPL (>= 3)

URL http://gforge.se

BugReports https://github.com/gforge/Gmisc/issues

Biarch yes

Imports grid, stringr, lattice, sp, utils, methods

Depends Hmisc

Suggests testthat, XML

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2014-07-28 08:09:01

R topics documented:

`Gmisc-package` *Collection of functions for plotting relations, generating tables, and more.*

Description

This is a collection of functions that I've found useful in my research. The package is inspired by Frank Harrell's **Hmisc** package. The main focus is on tables, plots, and **knitr**-integration.

Awesome tables

For tables you'll find the convenient `htmlTable` that I have used for advanced table layout. A major focus has been to have it compatible with LibreOffice (you can copy/past from there into word) as I generally want to be able to send my documents to a journal in .doc/.docx format. **Note:** it is now in RStudio possible to copy->paste directly from the viewer into a MS Word document with minimal layout loss.

The `getDescriptionStatsBy` is a straight forward function that aims at helping you to generate descriptive table stratified by different variables. In other words, the function returns everything you need for generating a *Table 1* ready for publication. This function is accompanied by the `describeMean`, `describeMedian`, `describeProp`, and `describeFactors` functions.

Convenient knitr-helpers

One of the main priorities of this package is to make the preparation of publication-ready manuscripts through the **knitr**-package. The `figCapNo` can be used for automated figure counting. The `pvalueFormatter` tries to simplify rounding of p-values, e.g. you may be ok with just 0.0005 as a p-value but when you come close to the "magic" 0.05 value you may want to have two significant digits, i.e. 0.048 instead of just 0.05. The `outputInt` simply transforms a large integer digit to proper formatting.

Some fancy plots

The forest plot function, `forestplot2`, is a more general version of the original **rmeta**-packages `forestplot` implementation. The aim is at using forest plots for more than just meta-analyses.

The transition plot function, `transitionPlot`, is for descriptive purposes. It tries to illustrate the size of change between one state and the next, i.e. a transition. This is basically a graph of based upon `table(var1, var2)`.

The Singular value decomposition is a common method for reducing the number of variables. Unfortunately this compression can reduce the interpretability of the model. The `getSvdMostInfluential` function tries to remedy that by identifying the most influential elements from the `V`-matrix.

The `getTicks` tries to format ticks for plots in a nicer way. The major use is for exponentials where ticks are generated using the 2^n since a doubling is a concept easy to grasp even for non-statisticians.

Other stuff

The `insertRowAndKeepAttr` simply adds a row while remembering all the attributes previously set by using the `copyAllNewAttributes`. The `mergeLists` tries to merge lists that do not have identical elements.

`bezierArrowGradient` *A bezier arrow with gradient*

Description

This is an experimental addition to the original `bezierArrowSmpl` with the addition of a gradient in the center of the arrow that fades.

Usage

```
bezierArrowGradient(width = 0.05, clr = "#000000", default.units = "npc",
  align_2_axis = TRUE, grdt_type = c("triangle"),
  grdt_decrease_prop = 0.4, grdt_start_prop = 0.4, grdt_clr_prop = 0.7,
  grdt_line_width = NA, grdt_clr = "#2F4F2F", vp = NULL, gp = gpar(),
  ...)
```

Arguments

width	The width of the arrow, either a numeric single number or a unit. **Note:** The arrow does not rely on lwd but on actual width. Same as in the bezierArrowSmpl
default.units	A string indicating the default units to use width is given as numeric vectors.
clr	The color of the arrow. This is the main color of the arrow and not the gradient color.
align_2_axis	Indicates if the arrow should be vertically/horizontally aligned. This is useful for instance if the arrow attaches to a box.
grdt_type	The type of growth and gradient that is to be used, currently it only supports triangle (I'm considering adding bezier curves but currently I'm a little tired of coding)
grdt_decrease_prop	The proportion of the full length that should be decreasing.
grdt_start_prop	The proportion of the full length that should be a constant color before decreasing.
grdt_clr_prop	The proportion of the gradient that should be decreasing. This is a proportion of the grdt_decrease_prop and the grdt_start_prop combined.
grdt_line_width	The width of the border line. If not specified it defaults to 5 % of the original width, note the gradient's width is thus 90 %.
grdt_clr	The color of the gradient. It is the color that transits into the clr of the arrow.
...	Passed on to bezierArrowSmpl
gp	An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
vp	A **grid** viewport object (or NULL).

Value

A grob of gList-type

Note

The triangle section of the arrow is not currently included in the gradient.

Examples

```
library(grid)
grid.newpage()
arrowGrob <- bezierArrowGradient(x = c(.1,.3,.6,.9),
                                 y = c(0.2, 0.2, 0.9, 0.9))
grid.draw(arrowGrob)
```

bezierArrowSmpl *A simple bezier arrow*

Description

This is an alternative to the grid packages bezierGrob with the advantage that it allows you to draw an arrow with a specific unit width. Note, it has only a end-arrow at this point.

Usage

```
bezierArrowSmpl(x = c(0.2, 0.7, 0.3, 0.9), y = c(0.2, 0.2, 0.9, 0.9),
  width = 0.05, clr = "#000000", default.units = "npc",
  arrow = list(base = unit(0.1, "npc"), length = unit(0.1, "npc")),
  align_2_axis = TRUE, name = NULL, gp = gpar(), vp = NULL)
```

Arguments

x	A numeric vector or unit object specifying x-locations of spline control points.
y	A numeric vector or unit object specifying y-locations of spline control points.
width	The width of the arrow, either a numeric single number or a unit. **Note:** The arrow does not rely on lwd but on actual width.
clr	The color of the arrow.
default.units	A string indicating the default units to use if x or y are only given as numeric vectors.
arrow	This is a list with all the **base** (width) and the desired **length** for the arrow. **Note:** This differs from the original bezierGrob function.
align_2_axis	Indicates if the arrow should be vertically/horizontally aligned. This is useful for instance if the arrow attaches to a box.
name	A character identifier.
gp	An object of class gpar, typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.
vp	A Grid viewport object (or NULL).

Value

A grob of the class polygonGrob with attributes that correspond to the bezier points.

Examples

```
library(grid)
grid.newpage()
arrowGrob <- bezierArrowSmpl(x = c(.1,.3,.6,.9),
                             y = c(0.2, 0.2, 0.9, 0.9))
grid.draw(arrowGrob)
```

copyAllNewAttributes *A simple thing to keep the attributes*

Description

Skips the attributes that the to object already has to avoid overwriting dim and other important attributes

Usage

```
copyAllNewAttributes(from, to, attr2skip = c(), attr2force = c())
```

Arguments

from	The from object
to	The to object
attr2skip	An optional lists of attributes that you may want to avoid having copied
attr2force	An optional lists of attributes that you may want to force copy even if they already exist in the new object

Value

object The to object

Examples

```
a <- "test"
attr(a, 'wow') <- 1000
b <- a
b <- copyAllNewAttributes(a, b)
print(attr(b, 'wow'))
```

describeFactors *A function that returns a description proportion that contains the number and the percentage*

Description

A function that returns a description proportion that contains the number and the percentage

Usage

```
describeFactors(x, html = FALSE, digits = 1, number_first = TRUE,
  show_missing = FALSE, show_missing_digits = digits,
  horizontal_proportions = NULL, percentage_sign = TRUE, language = "en",
  ...)
```

Arguments

x	The variable that you want the statistics for
digits	The number of decimals used for the percentage
html	If HTML compatible output should be used instead of default LaTeX
number_first	If the number should be given or if the percentage should be presented first. The second is encapsulated in parentheses ().
show_missing	This indicates if missing should be added as a separate row below all other.
show_missing_digits	The number of digits to use for the missing percentage, defaults to the overall digits.
horizontal_proportions	This is default NULL and indicates that the proportions are to be interpreted in a vertical manner. If we want the data to be horizontal, i.e. the total should be shown and then how these differ in the different groups then supply the function with the total number in each group, i.e. if done in a by manner as in getDescriptionStatsBy it needs to provide the number before the by() command.
percentage_sign	If you want to suppress the percentage sign you can set this variable to FALSE. You can also choose something else that the default % if you so wish by setting this variable.
language	The ISO-639-1 two-letter code for the language of interest. Currently only english is distinguished from the ISO format using a ',' as the separator in the outputInt function.
...	Passed on to outputInt

Value

A string formatted for printing either latex by HTML

See Also

getDescriptionStatsBy

Other description functions: describeMean; describeMedian; describeProp

Examples

```
set.seed(1)
describeFactors(sample(50, x=c("A","B", "C"), replace=TRUE))
describeFactors(sample(50, x=c("A","B", "C", NA), replace=TRUE), show_missing=TRUE)

n <- 500
my_var <- factor(sample(size=n, x=c("A","B", "C", NA), replace=TRUE))
my_exp <- rbinom(n=n, size=1, prob=0.2)
total <- table(my_var, useNA="ifany")
by(my_var,
   INDICES=my_exp,
```

```
  FUN=describeFactors,
  show_missing="ifany",
  horizontal_proportions = total)
```

`describeMean` *A function that returns a description mean that contains the standard deviation*

Description

A function that returns a description mean that contains the standard deviation

Usage

```
describeMean(x, html = FALSE, digits = 1, number_first = TRUE,
  show_missing = FALSE, show_missing_digits = digits,
  horizontal_proportions = NULL, percentage_sign = TRUE, plusmin_str,
  language = "en", ...)
```

Arguments

`x`	The variable that you want the statistics for
`digits`	The number of decimals used
`html`	If HTML compatible output shoudl be used instead of default LaTeX
`number_first`	If the number should be given or if the percentage should be presented first. The second is encapsulated in parentheses (). This is only used together with the show_missing variable.
`show_missing`	This indicates if missing should be added as a separate row below all other.
`show_missing_digits`	The number of digits to use for the missing percentage, defaults to the overall `digits`.
`horizontal_proportions`	Is only active if show_missing since this is the only case of a proportion among continuous variables. This is default NULL and indicates that the proportions are to be interpreted in a vertical manner. If we want the data to be horizontal, i.e. the total should be shown and then how these differ in the different groups then supply the function with the total number in each group, i.e. if done in a by manner as in `getDescriptionStatsBy` it needs to provide the number before the by() command. Note! This calls the `describeFactors` since the horizontal interpretation looses the vertical information in the second category and is thus better interpreted as a whole.
`percentage_sign`	If you want to suppress the percentage sign you can set this variable to FALSE. You can also choose something else that the default % if you so wish by setting this variable. Note, this is only used when combined with the missing information.

plusmin_str	Provide if you want anything other than the plus minus sign suited for the given output format.
language	The ISO-639-1 two-letter code for the language of interest. Currently only english is distinguished from the ISO format using a ',' as the separator in the outputInt function.
...	Passed on to describeFactors

Value

A string formatted for printing either latex by HTML

See Also

getDescriptionStatsBy

Other description functions: describeFactors; describeMedian; describeProp

Examples

```
describeMean(1:10)
describeMean(c(1:10, NA), show_missing=TRUE)
```

---

describeMedian *A function that returns a description median that contains the interquartile range or the full range*

Description

A function that returns a description median that contains the interquartile range or the full range

Usage

```
describeMedian(x, iqr = TRUE, html = FALSE, digits = 1,
  number_first = TRUE, show_missing = FALSE, show_missing_digits = digits,
  horizontal_proportions = NULL, percentage_sign = TRUE, language = "en",
  ...)
```

Arguments

x	The variable that you want the statistics for
iqr	If interquartile range should be used
digits	The number of decimals used
html	If HTML compatible output shoudl be used instead of default LaTeX
number_first	If the number should be given or if the percentage should be presented first. The second is encapsulated in parentheses (). This is only used together with the show_missing variable.

show_missing This indicates if missing should be added as a separate row below all other.

show_missing_digits
The number of digits to use for the missing percentage, defaults to the overall digits.

horizontal_proportions
Is only active if show_missing since this is the only case of a proportion among continuous variables. This is default NULL and indicates that the proportions are to be interpreted in a vertical manner. If we want the data to be horizontal, i.e. the total should be shown and then how these differ in the different groups then supply the function with the total number in each group, i.e. if done in a by manner as in getDescriptionStatsBy it needs to provide the number before the by() command. Note! This calls the describeFactors since the horizontal interpretation looses the vertical information in the second category and is thus better interpreted as a whole.

percentage_sign
If you want to suppress the percentage sign you can set this variable to FALSE. You can also choose something else that the default % if you so wish by setting this variable. Note, this is only used when combined with the missing information.

language The ISO-639-1 two-letter code for the language of interest. Currently only english is distinguished from the ISO format using a ',' as the separator in the outputInt function.

... Passed on to describeFactors

Value

A string formatted for printing either latex by HTML

See Also

getDescriptionStatsBy

Other description functions: describeFactors; describeMean; describeProp

Examples

```
describeMedian(1:10)
describeMedian(c(1:10, NA), show_missing=TRUE)
```

---

describeProp *A function that returns a description proportion that contains the number and the percentage*

Description

A function that returns a description proportion that contains the number and the percentage

Usage

```
describeProp(x, html = FALSE, digits = 1, number_first = TRUE,
  show_missing = FALSE, show_missing_digits = digits,
  horizontal_proportions = NULL, default_ref = "First",
  percentage_sign = TRUE, language = "en", ...)
```

Arguments

x	The variable that you want the statistics for
digits	The number of decimals used for the percentage
html	If HTML compatible output should be used instead of default LaTeX
number_first	If the number should be given or if the percentage should be presented first. The second is encapsulated in parentheses ().
show_missing	This indicates if missing should be added as a separate row below all other. This will always be converted into the describeFactor function if there is a missing row.
show_missing_digits	The number of digits to use for the missing percentage, defaults to the overall digits.
horizontal_proportions	This is default NULL and indicates that the proportions are to be interpreted in a vertical manner. If we want the data to be horizontal, i.e. the total should be shown and then how these differ in the different groups then supply the function with the total number in each group, i.e. if done in a by manner as in getDescriptionStatsBy it needs to provide the number before the by() command. Note! This calls the describeFactors since the horizontal interpretation looses the vertical information in the second category and is thus better interpreted as a whole.
default_ref	If you use proportions with only one variable it can be useful to set the reference level that is of interest to show. This can wither be "First", level name or level number.
percentage_sign	If you want to suppress the percentage sign you can set this variable to FALSE. You can also choose something else that the default % if you so wish by setting this variable.
language	The ISO-639-1 two-letter code for the language of interest. Currently only english is distinguished from the ISO format using a ',' as the separator in the outputInt function.
...	Passed on to outputInt

Value

A string formatted for printing either latex by HTML

See Also

getDescriptionStatsBy, describeFactors

Other description functions: describeFactors; describeMean; describeMedian

Examples

```
describeProp(factor(sample(50, x=c("A","B"), replace=TRUE)))
describeProp(factor(sample(50, x=c("A","B", NA), replace=TRUE)), show_missing=TRUE)
```

---

figCapNo *Adds a figure caption number*

Description

The function relies on `options("fig_caption_no")` in order to keep track of the last number. If you want to force the caption function to skip captions while still using it in the knitr fig.cap option then simply set `options(fig_caption_no = FALSE)`

Usage

```
figCapNo(str, roman = getOption("fig_caption_no_roman", FALSE),
  sprintf_str = getOption("fig_caption_no_sprintf", "Fig. %s: %s"))
```

Arguments

str	The string that is to be prepended with string
roman	Whether or not to use roman numbers instead of arabic. Can also be set through `options(fig_caption_no_roman = TRUE)`
sprintf_str	An sprintf formatted string where the first argument is reserved for the string generated by the counter and the second one is for the caption text. Can also be set through `options(fig_caption_no_sprintf = TRUE)`

See Also

Other figure caption functions: figCapNoLast; figCapNoNext

Examples

```
## Not run:
```{r, fig.cap=figCapNo("My nice plot")}
plot(1:10 + rnorm(10), 1:10)
```

End(Not run)
org_opts <- options(fig_caption_no = 2,
                    fig_caption_no_sprintf = "Figure %s: %s")
figCapNo("A plot with caption number = 3")
```

```
org_opts <- options(fig_caption_no = TRUE)
figCapNo("A plot with caption number =  1")

# Use default setting
options(fig_caption_no_sprintf = NULL)
figCapNo("A plot with caption number =  2")

# Return the original settings
options(org_opts)
```

`figCapNoLast` *Gets the last figure caption number*

Description

The function relies on `options("fig_caption_no")` in order to keep track of the last number.

Usage

```
figCapNoLast(roman = getOption("fig_caption_no_roman", FALSE))
```

Arguments

`roman`	Whether or not to use roman numbers instead of arabic. Can also be set through `options(fig_caption_no_roman = TRUE)`

See Also

Other figure caption functions: `figCapNoNext`; `figCapNo`

Examples

```
org_opts <- options(fig_caption_no=1)
figCapNoLast()
options(org_opts)
```

`figCapNoNext` *Gets the next figure caption number*

Description

The function relies on `options("fig_caption_no")` in order to keep track of the last number.

Usage

```
figCapNoNext(roman = getOption("fig_caption_no_roman", FALSE))
```

Arguments

roman	Whether or not to use roman numbers instead of arabic. Can also be set through `options(fig_caption_no_roman = TRUE)`

See Also

Other figure caption functions: `figCapNoLast`; `figCapNo`

Examples

```
org_opts <- options(fig_caption_no=1)
figCapNoLast()
options(org_opts)
```

---

forestplot2 *Create a forest plot*

Description

forestplot2 is based on the **rmeta** 2.16 `forestplot` function. This function resolves some limitations of the original functions such as:

- Adding expressions: Allows use of expressions, e.g. `expression(beta)`
- Multiple bands: Using multiple confidence bands for the same label
- Autosize: Adapts to windows size

Usage

```
forestplot2(labeltext, mean, lower, upper, align = NULL, is.summary = FALSE,
  fontfamily.summary = NULL, fontfamily.labelrow = NULL, clip = c(-Inf,
  Inf), xlab = "", zero = 0, graphwidth = "auto", lineheight = "auto",
  col = fpColors(), xlog = FALSE, xticks = NULL, xticks.digits = 2,
  lwd.xaxis = NULL, lwd.zero = NULL, lwd.ci = NULL, cex = 1,
  cex.axis = cex * 0.6, boxsize = NULL, mar = unit(rep(5, times = 4),
  "mm"), main = NULL, legend = NULL, legend.pos = "top",
  legend.cex = cex * 0.8, legend.gp = NULL, legend.r = unit(0, "snpc"),
  legend.padding = unit(ifelse(length(legend.gp) > 0, 3, 0), "mm"),
  legend.title = NULL, new_page = FALSE, confintNormalFn = fpDrawNormalCI,
  confintSummaryFn = fpDrawSummaryCI, legendMarkerFn = NULL, ...)
```

Arguments

labeltext	A list, matrix, vector or expression with the names of each row. The list should be wrapped in m x n number to resemble a matrix: `list(list("rowname 1 col 1", "rowname 2 col 1` You can also provide a matrix although this cannot have expressions by design: `matrix(c("rowname 1 col 1", "rowname 2 col 1", "r1c2", "beta"), ncol=2)` Use NA:s for blank spaces and if you provide a full column with NA then that column is a empty column that adds some space.

`mean` A vector or a matrix with the averages

`lower` The lower bound of the confidence interval for the forestplot, needs to be the same format as the mean, i.e. matrix/vector of equal columns & length

`upper` The upper bound of the confidence interval for the forestplot, needs to be the same format as the mean, i.e. matrix/vector of equal columns & length

`align` Vector giving alignment (l,r,c) for columns of table

`is.summary` A vector indicating by TRUE/FALSE if the value is a summary value which means that it will have a different font-style.

`fontfamily.summary`
The fontfamily of the summary

`fontfamily.labelrow`
The fontfamily of a regular row

`clip` Lower and upper limits for clipping confidence intervals to arrows

`xlab` x-axis label

`zero` x-axis coordinate for zero line

`graphwidth` Width of confidence interval graph, see `unit` for details on how to utilize mm etc. The default is `auto`, that is it uses up whatever space that is left after adjusting for text size and legend.

`lineheight` Height of the graph. By default this is `auto` and adjustes to the space that is left after adjusting for x-axis size and legend. Sometimes it might be desireable to set the line height to a certain height, for instance if you have several forestplots you may want to standardize their line height, then you set this variable to a certain height, note this should be provided as a `unit` object. A good option is to set the line height to `unit(2, "cm")`. A third option is to set line height to "lines" and then you get 50 line height.

`col` See `fpColors`

`xlog` If TRUE, x-axis tick marks are exponentiated

`xticks` Optional user-specified x-axis tick marks. Specify NULL to use the defaults, numeric(0) to omit the x-axis.

`xticks.digits` The number of digits to allow in the x-axis if this is created by default.

`lwd.xaxis` lwd for the xaxis

`lwd.zero` lwd for the vertical line that gives the no-effect line

`lwd.ci` lwd for the confidence bands

`cex` The font adjustment

`cex.axis` The font adjustment for the x-xaxis, defaults to 60 % of the cex parameter.

`boxsize` Override the default box size based on precision

`mar` A numerical vector of the form c(bottom, left, top, right) of the type `unit()`

`main` The title of the plot if any, default `NULL`

`legend` Legen corresponding to the number of bars.

`legend.pos` The position of the legend, either at the "top" or the "right" unlesss positioned inside the plot. If you want the legend to be positioned inside the plot then you have to provide a list with the same x & y qualities as `legend`. For instance if you want the legend to be positioned at the top right corner then use `legend.pos = list("topright")` - this is equivalent to `legend.pos = list(x=1, y=1)`. If you want to have a distance from the edge of the graph then add a inset to the list, e.g. `legend.pos = list("topright", "inset"=.1)` - the inset should be either a `unit` element or a value between 0 and 1. The default is to have the boxes aligned vertical, if you want them to be in a line then you can specify the "align" option, e.g. `legend.pos = list("topright", "inset"=.1, "align"="horizontal")`

`legend.cex` The cex size of the legend, defaults to 80 % procent of cex parameter.

`legend.gp` The `gpar` options for the legend. If you want the background color to be light grey then use `legend.gp = gpar(fill = "lightgrey")`. If you want a border then set the col argument: `legend.gp = gpar(fill = "lightgrey", col="black")`. You can also use the lwd and lty argument as usual, `legend.gp = gpar(lwd=2, lty=1)`, will result in a black border box of line type 1 and line width 2.

`legend.r` The box can have rounded edges, check out `grid.roundrect`. The r option should be a `unit` object. This is by default `unit(0, "snpc")` but you can choose any value that you want. The `"snpc"` unit is the preferred option.

`legend.padding` The padding for the legend box, only used if box is drawn. This is the distance from the border to the text/boxes of the legend.

`legend.title` The title of the legend if any, default to NULL

`new_page` If you want the plot to appear on a new blank page then set this to `TRUE`, by default it is `FALSE`.

`confintNormalFn`
You can specify exactly how the line with the box is drawn for the normal (i.e. non-summary) confidence interval by changing this parameter to your own function or some of the alternatives provided in the package. It defaults to the box function `fpDrawNormalCI`.

`confintSummaryFn`
Same as previous argument but for the summary outputs and it defaults to `fpDrawSummaryCI`.

`legendMarkerFn` What type of function should be used for drawing the legends, this can be a list if you want different functions. It defaults to a box if you have anything else than a single function or the number of columns in the `mean` argument.

`...` Passed on to the `confintNormalFn` and `confintSummaryFn` arguments

Details

Using multiple bands for the same label can be interesting when one wants to compare different outcomer. It can also be an alternative when you want to show both crude and adjusted estimates.

Known issues: the x-axis does not entirely respect the margin. Autosizing boxes is not always the best option, try to set these manually as much as possible.

Value

void

Author(s)

Max Gordon, Thomas Lumley

Examples

```
###############################################################
# Simple examples of how to do a forestplot with forestplot2 #
###############################################################

ask <- par(ask=TRUE)
library(grid)

# A basic example, create some fake data
row_names <- list(list("test = 1", expression(test >= 2)))
test_data <- data.frame(coef=c(1.59, 1.24),
  low=c(1.4, 0.78),
  high=c(1.8, 1.55))
forestplot2(row_names,
            test_data$coef,
            test_data$low,
            test_data$high,
            zero = 1,
            cex  = 2,
            lineheight = "auto",
            xlab = "Lab axis txt",
            new_page = TRUE)

# Print two plots side by side using the grid
# package's layout option for viewports
grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
pushViewport(viewport(layout.pos.col = 1))
forestplot2(row_names,
            test_data$coef,
            test_data$low,
            test_data$high,
            zero = 1,
            cex  = 2,
            lineheight = "auto",
            xlab = "Lab axis txt")
popViewport()
pushViewport(viewport(layout.pos.col = 2))
forestplot2(row_names,
            test_data$coef,
            test_data$low,
            test_data$high,
            zero = 1,
            cex  = 2,
            lineheight = "auto",
            xlab = "Lab axis txt")
popViewport()
popViewport()
```

```
# An advanced test
test_data <- data.frame(coef1=c(1, 1.59, 1.3, 1.24),
  coef2=c(1, 1.7, 1.4, 1.04),
  low1=c(1, 1.3, 1.1, 0.99),
  low2=c(1, 1.6, 1.2, 0.7),
  high1=c(1, 1.94, 1.6, 1.55),
  high2=c(1, 1.8, 1.55, 1.33))

col_no <- grep("coef", colnames(test_data))
row_names <- list(
  list("Category 1", "Category 2", "Category 3", expression(Category >= 4)),
  list("ref",
    substitute(expression(bar(x) == val),
               list(val = round(rowMeans(test_data[2, col_no]), 2))),
    substitute(expression(bar(x) == val),
               list(val = round(rowMeans(test_data[3, col_no]), 2))),
    substitute(expression(bar(x) == val),
               list(val = round(rowMeans(test_data[4, col_no]), 2))))
)

coef <- with(test_data, cbind(coef1, coef2))
low <- with(test_data, cbind(low1, low2))
high <- with(test_data, cbind(high1, high2))
forestplot2(row_names, coef, low, high,
            main="Cool study",
            zero = 1, boxsize=0.5,
            col=fpColors(box=c("royalblue", "gold"),
                         line=c("darkblue", "orange"),
                         summary=c("darkblue", "red")),
            xlab="The estimates",
            new_page = TRUE,
            legend.title="Group",
            legend=c("Treatment", "Placebo"),
            legend.pos=list("topright"),
            legend.r = unit(.1, "snpc"),
            legend.gp = gpar(col="#CCCCCC", lwd=1.5))

# An example of how the exponential works
test_data <- data.frame(coef=c(2.45, 0.43),
  low=c(1.5, 0.25),
  high=c(4, 0.75),
  boxsize=c(0.5, 0.5))
row_names <- cbind(c("Name", "Variable A", "Variable B"),
                   c("HR", test_data$coef))
test_data <- rbind(rep(NA, 3), test_data)

forestplot2(labeltext = row_names,
            is.summary=c(TRUE, FALSE, FALSE),
            mean      = test_data$coef,
            lower     = test_data$low,
            upper     = test_data$high,
```

```
            boxsize   = test_data$boxsize,
            zero      = 1,
            xlog      = TRUE,
            col = fpColors(lines="red", box="darkred"),
            new_page = TRUE)


par(ask=ask)
```

`fpColors` *A function for the color elements used in the forestplot2()*

Description

This function encapsulates all the colors that are used in the `forestplot2` function. As there are plenty of color options this function gathers them all in one place.

Usage

```
fpColors(all.elements, box = "black", lines = "gray", summary = "black",
  zero = "lightgray", text = "black", axes = "black")
```

Arguments

`all.elements`	A color for all the elements. If set to NULL then it's set to the par("fg") color
`box`	The color of the box indicating the estimate
`lines`	The color of the confidence lines
`summary`	The color of the summary
`zero`	The color of the zero line
`text`	The color of the text
`axes`	The color of the x-axis at the bottom

Details

If you have several values per row in a forestplot you can set a color to a vector where the first value represents the first line/box, second the second line/box etc. The vectors are only valid for the box & `lines` options.

This function is a copy of the `meta.colors` function in the **rmeta** package.

Value

list A list with the elements:

`box`	the color of the box/marker
`lines`	the color of the lines
`summary`	the color of the summary

zero	the color of the zero vertical line
text	the color of the text
axes	the color of the axes

Author(s)

Max Gordon, Thomas Lumley

See Also

Other forestplot functions: fpDrawCircleCI, fpDrawDiamondCI, fpDrawNormalCI, fpDrawPointCI, fpDrawSummaryCI

---

fpDrawNormalCI *Draw standard confidence intervals*

Description

A function that is used to draw the different confidence intervals for the non-summary lines. Use the fpDrawNormalCI function as a template if you want to make your own funky line + marker.

Usage

```
fpDrawNormalCI(lower_limit, estimate, upper_limit, size, y.offset = 0.5,
  clr.line, clr.marker, lwd, ...)

fpDrawDiamondCI(lower_limit, estimate, upper_limit, size, y.offset = 0.5,
  clr.line, clr.marker, lwd, ...)

fpDrawCircleCI(lower_limit, estimate, upper_limit, size, y.offset = 0.5,
  clr.line, clr.marker, lwd, ...)

fpDrawPointCI(lower_limit, estimate, upper_limit, size, y.offset = 0.5,
  clr.line, clr.marker, lwd, pch = 1, ...)

fpDrawSummaryCI(lower_limit, estimate, upper_limit, size, col, y.offset = 0.5,
  ...)
```

Arguments

lower_limit	The lower limit of the confidence line. A native numeric variable that can actually be outside the boundaries. If you want to see if it is outside then convert it to 'npc' and see if the value ends up more than 1 or less than 0. Here's how you do the conversion: `convertX(unit(upper_limit, "native"), "npc", valueOnly = TRUE)` and the convertX together with unit is needed to get the right values while you need to provide the valueOnly as you cannot compare a unit object.

estimate	The estimate indicating the placement of the actual box. Note, this can also be outside bounds and is provided in a numeric format the same way as the lower_limit.
upper_limit	The upper limit of the confidence line. See lower_limit for details.
size	The actual size of the box/diamond/marker. This provided in the 'snpc' format to generate a perfect marker. Although you can provide it alternative units as well, this is useful for the legends to work nicely.
y.offset	If you have multiple lines they need an offset in the y-direction.
clr.line	The color of the line.
clr.marker	The color of the estimate marker
lwd	Line width
...	Allows additional parameters for sibling functions
pch	Type of point see grid.points for details
col	The color of the summary diamond.

Value

void The function outputs the line using grid compatible functions and does not return anything.

See Also

forestplot2

Other forestplot functions: fpColors

Examples

```
ask <- par(ask=TRUE)
library(grid)
test_data <- data.frame(coef1=c(1, 1.59, 1.3, 1.24),
                        coef2=c(1, 1.7, 1.4, 1.04))

test_data$low1 <- test_data$coef1 - 1.96*c(0, .2, .1, .15)
test_data$high1 <- test_data$coef1 + 1.96*c(0, .2, .1, .15)

test_data$low2 <- test_data$coef2 - 1.96*c(0, .1, .15, .2)
test_data$high2 <- test_data$coef2 + 1.96*c(0, .1, .15, .2)

col_no <- grep("coef", colnames(test_data))
row_names <- list(
  list("Category 1", "Category 2", "Category 3", expression(Category >= 4)),
  list("ref",
       substitute(expression(bar(x) == val),
                  list(val = round(rowMeans(test_data[2, col_no]), 2))),
       substitute(expression(bar(x) == val),
                  list(val = round(rowMeans(test_data[3, col_no]), 2))),
       substitute(expression(bar(x) == val),
                  list(val = round(rowMeans(test_data[4, col_no]), 2))))
)
```

```
coef <- with(test_data, cbind(coef1, coef2))
low <- with(test_data, cbind(low1, low2))
high <- with(test_data, cbind(high1, high2))

# Change all to diamonds
forestplot2(row_names, coef, low, high,
            confintNormalFn=fpDrawDiamondCI,
            main="Cool study",
            zero = 1, boxsize=0.5,
            col=fpColors(box=c("royalblue", "gold"),
                         line=c("darkblue", "orange"),
                         summary=c("darkblue", "red")),
            xlab="The estimates",
            new_page = TRUE,
            legend.title="Group",
            legend=c("Treatment", "Placebo"),
            legend.pos=list("topright"),
            legend.r = unit(.1, "snpc"),
            legend.gp = gpar(col="#CCCCCC", lwd=1.5))

# Change first to diamonds
forestplot2(row_names, coef, low, high,
            confintNormalFn=c("fpDrawDiamondCI",
                              rep("fpDrawNormalCI",
                                  times=nrow(coef)-1)),
            main="Cool study",
            zero = 1, boxsize=0.5,
            col=fpColors(box=c("royalblue", "gold"),
                         line=c("darkblue", "orange"),
                         summary=c("darkblue", "red")),
            xlab="The estimates",
            new_page = TRUE,
            legend.title="Group",
            legend=c("Treatment", "Placebo"),
            legend.pos=list("topright"),
            legend.r = unit(.1, "snpc"),
            legend.gp = gpar(col="#CCCCCC", lwd=1.5))

# You can also use a list with the actual functions
# as long as it is formatted [[row]][[column]]
# Note: if you have a non-square input then
# the software will reformat [[col]][[row]]
# to [[row]][[col]]
forestplot2(row_names, coef, low, high,
            confintNormalFn=list(list(fpDrawDiamondCI, fpDrawCircleCI),
                                 list(fpDrawNormalCI, fpDrawNormalCI),
                                 list(fpDrawNormalCI, fpDrawCircleCI),
                                 list(fpDrawNormalCI, fpDrawNormalCI)),
            main="Cool study",
            zero = 1, boxsize=0.5,
            col=fpColors(box=c("royalblue", "gold"),
                         line=c("darkblue", "orange"),
```

```
                    summary=c("darkblue", "red")),
             xlab="The estimates",
             new_page = TRUE,
             legend.title="Group",
             legend=c("Treatment", "Placebo"),
             legend.pos=list("topright"),
             legend.r = unit(.1, "snpc"),
             legend.gp = gpar(col="#CCCCCC", lwd=1.5))

par(ask=ask)
```

`getDescriptionStatsBy` *Creating of description statistics*

Description

A function that returns a description statistic that can be used for creating a publication "table 1" when you want it by groups. The function identifies if the variable is a continuous, binary or a factored variable. The format is inspired by NEJM, Lancet & BMJ.

Usage

```
getDescriptionStatsBy(x, by, digits = 1, html = FALSE, NEJMstyle = FALSE,
  numbers_first = TRUE, statistics = FALSE, sig.limit = 10^-4,
  two_dec.limit = 10^-2, show_missing = FALSE,
  show_missing_digits = digits, continuous_fn = describeMean,
  prop_fn = describeProp, factor_fn = describeFactors,
  show_all_values = FALSE, hrzl_prop = FALSE, add_total_col,
  total_col_show_perc = TRUE, use_units = FALSE, default_ref = "First",
  percentage_sign = TRUE)
```

Arguments

`x`	The variable that you want the statistics for
`by`	The variable that you want to split into different columns
`digits`	The number of decimals used
`html`	If HTML compatible output shoudl be used instead of default LaTeX
`NEJMstyle`	Adds - no (%) at the end to proportions
`numbers_first`	If the number should be given or if the percentage should be presented first. The second is encapsulated in parentheses ().
`show_missing`	Show the missing values. This adds another row if there are any missing values.
`show_missing_digits`	The number of digits to use for the missing percentage, defaults to the overall `digits`.
`continuous_fn`	The method to describe continuous variables. The default is `describeMean`.

prop_fn
: The method used to describe proportions, see describeProp.

factor_fn
: The method used to describe factors, see describeFactors.

statistics
: Add statistics, fisher test for proportions and Wilcoxon for continuous variables

two_dec.limit
: The limit for showing two decimals .

sig.limit
: The significance limit for < sign, i.e. p-value 0.0000312 should be < 0.0001 with the default setting.

show_all_values
: This is by default false as for instance if there is no missing and there is only one variable then it is most sane to only show one option as the other one will just be a complement to the first. For instance sex - if you know gender then automatically you know the distribution of the other sex as it's 100 % - other %. To choose which one you want to show then set the default_ref parameter.

hrzl_prop
: This is default FALSE and indicates that the proportions are to be interpreted in a vertical manner. If we want the data to be horizontal, i.e. the total should be shown and then how these differ in the different groups then set this to TRUE.

add_total_col
: This adds a total column to the resulting table. You can also specify if you want the total column "first" or "last" in the column order.

total_col_show_perc
: This is by default true but if requested the percentages are surpressed as this sometimes may be confusing.

use_units
: If the Hmisc package's units() function has been employed it may be interesting to have a column at the far right that indicates the unit measurement. If this column is specified then the total column will appear before the units (if specified as last).

default_ref
: If you use proportions with only one variable, i.e. not show_all_values, then it can be useful to set the reference level that is of interest to show. This can wither be "First", level name or level number.

percentage_sign
: If you want to surpress the percentage sign you can set this variable to FALSE. You can also choose something else that the default % if you so wish by setting this variable.

Value

Returns a vector if vars wasn't specified and it's a continuous or binary statistic. If vars was a matrix then it appends the result to the end of that matrix. If the x variable is a factor then it does not append and you get a warning.

See Also

describeMean, describeProp, describeFactors, htmlTable

Examples

```
data(mtcars)
```

```
label(mtcars$mpg) <- "Gas"
units(mtcars$mpg) <- "Miles/(US) gallon"

label(mtcars$wt) <- "Weight"
units(mtcars$wt) <- "10<sup>3</sup> kg" # not sure the unit is correct

mtcars$am <- factor(mtcars$am, levels=0:1, labels=c("Automatic", "Manual"))
label(mtcars$am) <- "Transmission"

mtcars$gear <- factor(mtcars$gear)
label(mtcars$gear) <- "Gears"

# Make up some data for making it slightly more interesting
mtcars$col <- factor(sample(c("red", "black", "silver"),
                    size=NROW(mtcars), replace=TRUE))
label(mtcars$col) <- "Car color"

mpg_data <- getDescriptionStatsBy(mtcars$mpg, mtcars$am,
                                  use_units = TRUE,
                                  html = TRUE)
wt_data <- getDescriptionStatsBy(mtcars$wt, mtcars$am,
                                 use_units = TRUE,
                                 html = TRUE)

htmlTable(
  rbind(mpg_data, wt_data),
  caption  = "Continuous & binary variables",
  headings = c(sprintf("%s (SD)", levels(mtcars$am)), "Units"),
  rowlabel = "Variable",
  ctable   = TRUE)

gear_data <- getDescriptionStatsBy(mtcars$gear, mtcars$am)
col_data <- getDescriptionStatsBy(mtcars$col, mtcars$am)

htmlTable(rbind(gear_data, col_data),
  caption  = "Factored variables",
  colheads = sprintf("%s (%%)", levels(mtcars$am)),
  rowlabel = "Variable",
  rgroup   = c(label(gear_data),
               label(col_data)),
  n.rgroup = c(NROW(gear_data),
               NROW(col_data)),
  ctable   = TRUE)

# A little more advanced
mtcars$mpg[sample(1:NROW(mtcars), size=4)] <- NA
getDescriptionStatsBy(mtcars$mpg, mtcars$am, statistics=TRUE,
                      show_missing=TRUE)

# Do the horizontal version
getDescriptionStatsBy(mtcars$col, mtcars$am, statistics=TRUE,
                      show_missing=TRUE, hrzl_prop = TRUE)
```

```
mtcars$wt_with_missing <- mtcars$wt
mtcars$wt_with_missing[sample(1:NROW(mtcars), size=8)] <- NA
getDescriptionStatsBy(mtcars$wt_with_missing, mtcars$am, statistics=TRUE,
                      show_missing=TRUE, hrzl_prop = TRUE, total_col_show_perc = FALSE)


mtcars$col_with_missing <- mtcars$col
mtcars$col_with_missing[sample(1:NROW(mtcars), size=5)] <- NA
getDescriptionStatsBy(mtcars$col_with_missing, mtcars$am, statistics=TRUE,
                      show_missing=TRUE, hrzl_prop = TRUE, total_col_show_perc = FALSE)
```

`getSvdMostInfluential` *Gets the maximum contributor variables from svd()*

Description

This function is inspired by Jeff Leeks Data Analysis course where he suggests that one way to use the svd is to look at the most influential rows for first columns in the V matrix.

Usage

```
getSvdMostInfluential(mtrx, quantile, similarity_threshold,
  plot_selection = TRUE, plot_threshold = 0.05, varnames = NULL)
```

Arguments

`mtrx`	A matrix or data frame with the variables. Note: if it contains missing variables make sure to impute prior to this function as the svd can't handle missing values.
`quantile`	The SVD D-matrix gives an estimate for the amount that is explained. This parameter applies is used for selecting the columns that have that quantile of explanation.
`similarity_threshold`	A quantile for how close other variables have to be in value to maximum contributor of that particular column. If you only want the maximum value then set this value to 1.
`plot_selection`	As this is all about variable exploring it is often interesting to see how the variables were distributed among the vectors
`plot_threshold`	The threshold of the plotted bars, measured as percent explained by the D-matrix. By default it is set to 0.05.
`varnames`	A vector with alternative names to the colnames

Details

This function expands on that idea and adds the option of choosing more than just the most contributing variable for each row. For instance two variables may have a major impact on a certain component where the second variable has 95 probably important in that particular component it makes sense to include it in the selection.

It is of course useful when you have many continuous variables and you want to determine a subgroup to look at, i.e. finding the needle in the haystack.

Value

Returns a list with vector with the column numbers that were picked in the "most_influential" variable and the svd caluclation in the "svd"

Examples

```
org_par <- par(ask=TRUE)
set.seed(12345);

# Simulate data with a pattern
dataMatrix <- matrix(rnorm(15*160),ncol=15)
colnames(dataMatrix) <-
  c(paste("Pos.3:", 1:3, sep=" #"),
    paste("Neg.Decr:", 4:6, sep=" #"),
    paste("No pattern:", 7:8, sep=" #"),
    paste("Pos.Incr:", 9:11, sep=" #"),
    paste("No pattern:", 12:15, sep=" #"))
for(i in 1:nrow(dataMatrix)){
  # flip a coin
  coinFlip1 <- rbinom(1,size=1,prob=0.5)
  coinFlip2 <- rbinom(1,size=1,prob=0.5)
  coinFlip3 <- rbinom(1,size=1,prob=0.5)

  # if coin is heads add a common pattern to that row
  if(coinFlip1){
    cols <- grep("Pos.3", colnames(dataMatrix))
    dataMatrix[i, cols] <- dataMatrix[i, cols] + 3
  }

  if(coinFlip2){
    cols <- grep("Neg.Decr", colnames(dataMatrix))
   dataMatrix[i, cols] <- dataMatrix[i, cols] - seq(from=5, to=15, length.out=length(cols))
  }

  if(coinFlip3){
    cols <- grep("Pos.Incr", colnames(dataMatrix))
   dataMatrix[i,cols] <- dataMatrix[i,cols] + seq(from=3, to=15, length.out=length(cols))
  }
}

# Illustrate data
heatmap(dataMatrix, Colv=NA, Rowv=NA, margins=c(7,2), labRow="")
```

```
svd_out <- svd(scale(dataMatrix))

library(lattice)
b_clr <- c("steelblue", "darkred")
key <- simpleKey(rectangles = TRUE, space = "top", points=FALSE,
  text=c("Positive", "Negative"))
key$rectangles$col <- b_clr

b1 <- barchart(as.table(svd_out$v[,1]),
  main="First column",
  horizontal=FALSE, col=ifelse(svd_out$v[,1] > 0,
      b_clr[1], b_clr[2]),
  ylab="Impact value",
  scales=list(x=list(rot=55, labels=colnames(dataMatrix), cex=1.1)),
  key = key)

b2 <- barchart(as.table(svd_out$v[,2]),
  main="Second column",
  horizontal=FALSE, col=ifelse(svd_out$v[,2] > 0,
      b_clr[1], b_clr[2]),
  ylab="Impact value",
  scales=list(x=list(rot=55, labels=colnames(dataMatrix), cex=1.1)),
  key = key)

b3 <- barchart(as.table(svd_out$v[,3]),
  main="Third column",
  horizontal=FALSE, col=ifelse(svd_out$v[,3] > 0,
      b_clr[1], b_clr[2]),
  ylab="Impact value",
  scales=list(x=list(rot=55, labels=colnames(dataMatrix), cex=1.1)),
  key = key)

b4 <- barchart(as.table(svd_out$v[,4]),
  main="Fourth column",
  horizontal=FALSE, col=ifelse(svd_out$v[,4] > 0,
      b_clr[1], b_clr[2]),
  ylab="Impact value",
  scales=list(x=list(rot=55, labels=colnames(dataMatrix), cex=1.1)),
  key = key)

# Note that the fourth has the no pattern columns as the
# chosen pattern, probably partly because of the previous
# patterns already had been identified
print(b1, position=c(0,0.5,.5,1), more=TRUE)
print(b2, position=c(0.5,0.5,1,1), more=TRUE)
print(b3, position=c(0,0,.5,.5), more=TRUE)
print(b4, position=c(0.5,0,1,.5))

# Let's look at how well the SVD identifies
# the most influential columns
getSvdMostInfluential(dataMatrix,
                      quantile=.8,
```

```
                      similarity_threshold = .9,
                      plot_threshold = .05,
                      plot_selection = TRUE)
par(org_par)
```

---

getTicks *Ticks for plot axis*

Description

Gets the ticks in a formatted version. This is since I'm not always that fond of just pretty(1:10/5). In exponential form the ticks are determined from the 2-base, meaning that you get an intuitive feeling for when the value is doubled.

Usage

```
getTicks(low, high = low, clip = c(-Inf, Inf), exp = FALSE, digits = 0)
```

Arguments

low	lower bound, can be a single number or a vector
high	upper bound - optional, you can just have all data in the low variable
clip	if the ci are clipped
exp	If the value should be in exponential form (default)
digits	Number of digits - used in exp mode

Details

This function is far from perfect and I recommend specifying yourself the ticks that you want.

Value

Returns a vector with the ticks appropriate

Examples

```
test_data <- data.frame(coef=c(2, 0.5),
  low=c(1.5, 0.05),
  high=c(3, 0.75),
  boxsize=c(0.5, 0.5))

# Exponential form where the exponent base i 2 for easier understanding
getTicks(low = test_data$low,
  high = test_data$high,
  clip=c(-Inf, Inf),
  exp=TRUE)

# Non exponential form with using pretty
```

```
getTicks(low = test_data$low,
  high = test_data$high,
  clip=c(-Inf, Inf),
  exp=FALSE)


# A very simple example
getTicks(1:5*2.33,
  exp=FALSE)

# A slightly more advanced exponential version
getTicks(1:10*.33,
  digits=2,
  exp=TRUE)
```

`insertRowAndKeepAttr` *Insert a row into a matrix*

Description

Inserts a row and keeps the attributes copyAllNewAttributes

Usage

```
insertRowAndKeepAttr(m, r, v = NA, rName = "")
```

Arguments

m	matrix.
r	row number where the new row should be inserted
v	optional values for the new row.
rName	optional character string: the name of the new row.

Value

a matrix with one more row than the provided matrix m.

Author(s)

Max Gordon, Arne Henningsen

Examples

```
test <- matrix(1:4, ncol=2)
attr(test, 'wow') <- 1000
test <- insertRowAndKeepAttr(test, 2)
print(attr(test, 'wow'))
```

mergeLists *Merging of multiple lists*

Description

The merge allows for a recursive component where the lists are compared on the subelement. If one does not contain that element it will get NA in for those parameters.

Usage

```
mergeLists(..., lapplyOutput = NULL)
```

Arguments

...	Any number of lists that you want to merge
lapplyOutput	The lapply function outputs a number of lists and this is for specifically merging all of those.

Value

Returns a list with all the given lists.

Examples

```
v1 <- list("a"=c(1,2), b="test 1", sublist=list(one=20:21, two=21:22))
v2 <- list("a"=c(3,4), b="test 2", sublist=list(one=10:11, two=11:12, three=1:2))
mergeLists(v1, v2)
```

outputInt *SI or English formatting of an integer*

Description

English uses ',' between every 3 numbers while the SI format recommends a ' ' if x > 10^4. The scientific form 10e+? is furthermore avoided.

Usage

```
outputInt(x, language = "en", html = TRUE, ...)
```

Arguments

x	The integer variable
language	The ISO-639-1 two-letter code for the language of interest. Currently only english is distinguished from the ISO format using a ',' as the separator.
html	If the format is used in html context then the space should be a non-breaking space,
...	Passed to format

Value

string

Examples

```
outputInt(123456)
```

`pvalueFormatter` *Formats the p-values*

Description

Gets formatted p-values. For instance you often want 0.1234 to be 0.12 while also having two values up until a limit, i.e. 0.01234 should be 0.012 while 0.001234 should be 0.001. Furthermore you want to have < 0.001 as it becomes ridiculous to report anything below that value.

Usage

```
pvalueFormatter(pvalues, two_dec.limit = 10^-2, sig.limit = 10^-4,
  html = TRUE)
```

Arguments

`pvalues`	The p-values
`two_dec.limit`	The limit for showing two decimals
`sig.limit`	The significance limit for < sign
`html`	If the less than sign should be < or < as needed for html output.

Value

vector

Examples

```
pvalueFormatter(c(0.10234,0.010234, 0.0010234, 0.000010234))
```

splitLines4Table *A helper function for html/LaTeX line formatting*

Description

This function helps you to do a multiline table header in both html and in LaTeX. In html this isn't that tricky, you just use the
 command but in LaTeX I often find myself writing vbox/hbox stuff and therefore I've created this simple helper function

Usage

```
splitLines4Table(..., html = FALSE)
```

Arguments

...	The lines that you want to be joined
html	If it's suppose to be in html or LaTeX. Default is LaTeX.

Value

string

Examples

```
splitLines4Table("hello", "world")
splitLines4Table("hello", "world", html=TRUE)
splitLines4Table("hello", "world", list("A list", "is OK"))
```

transitionPlot *A transition plot*

Description

This plot purpose is to illustrate how states change before and after. In my research I use it before surgery and after surgery but it can be used in any situation where you have a change from one state to another

Usage

```
transitionPlot(transition_flow, type_of_arrow = c("grid", "simple",
  "gradient"), box_txt = rownames(transition_flow), tot_spacing = 0.2,
  box_width = 1/4, fill_start_box = "darkgreen", txt_start_clr = "white",
  fill_end_box = fill_start_box, txt_end_clr = txt_start_clr, cex = 2,
  min_lwd = if (type_of_arrow == "grid") 1 else unit(1, "mm"), max_lwd = if
  (type_of_arrow == "grid") 6 else unit(5, "mm"), lwd_prop_total = TRUE,
  arrow_clr = "#000000", abs_arrow_width = FALSE,
```

```
overlap_bg_clr = "#FFFFFF", overlap_order = 1:nrow(transition_flow),
overlap_add_width = if (type_of_arrow == "grid") 1.5 else unit(1, "mm"),
box_prop, mar = unit(rep(3, times = 4), "mm"), main = NULL,
box_label = NULL, box_label_pos = "top", box_label_cex = cex,
color_bar = TRUE, color_bar_cex = cex * 0.33, color_bar_labels,
color_bar_subspace, new_page = FALSE)
```

Arguments

transition_flow	This should be a matrix with the size of the transitions. The unit for each cell should be number of observations, row/column-proportions will show incorrect sizes. The matrix needs to be square. The best way to generate this matrix is probably just do a `table(starting_state, end_state)`. The rows represent the starting positions, while the columns the end positions. I.e. the first rows third column is the number of observations that go from the first class to the third class.
type_of_arrow	The types of arrow may be grid, simple, or gradient. Simple grid arrows are the bezierGrob arrows (not that pretty), simple is the bezierArrowSmpl that I've created to get a more exact control of the arrow position and width, while gradient corresponds to bezierArrowGradient allowing the arrow to have a fill color that slowly turns into the color of the arrow.
box_txt	The text to appear inside of the boxes. If you need line breaks then you need to manually add a \n inside the string.
tot_spacing	The proportion of the vertical space that is to be left empty. It is then split evenly between the boxes.
box_width	The width of the box. By default the box is one fourth of the plot width.
fill_start_box	The fill color of the start boxes. This can either be a single value ore a vector if you desire different colors for each box. If you specify box_prop then this has to be a 2 column matrix.
txt_start_clr	The text color of the start boxes. This can either be a single value ore a vector if you desire different colors for each box. If you specify box_prop then this has to be a 2 column matrix.
fill_end_box	The fill color of the end boxes. This can either be a single value ore a vector if you desire different colors for each box. If you specify box_prop then this has to be a 2 column matrix.
txt_end_clr	The text color of the end boxes. This can either be a single value ore a vector if you desire different colors for each box. If you specify box_prop then this has to be a 2 column matrix.
cex	The cex gpar of the text
min_lwd	The minimum width of the line that we want to illustrate the tranisition with.
max_lwd	The maximum width of the line that we want to illustrate the tranisition with.
lwd_prop_total	The width of the lines may be proportional to either the other flows from that box, or they may be related to all flows. This is a boolean parameter that is set to true by default, i.e. relating to all flows.

arrow_clr The color of the arrows. Usually black, can be a vector indicating each arrow from first to last arrow (counting from the top). If the vector is of the same length as the boxes then all box arrows will have the same color (that is all the arrows stemming from the left boxes)

abs_arrow_width
The width can either be absolute, i.e. each arrow headed for a box has the exact same width. The alternative is that the width is related to the line width.

overlap_bg_clr In order to enhance the 3D perspective and to make it easier to follow arrows the arrows have a background color to separate them from those underneath.

overlap_order The order from first->last for the lines. This means that the last line will be on top while the first one will appear at the bottom. This should be provided as a vector.

overlap_add_width
The width of the white cross-over line. You can specify this as a scalar multiplication of the current line width. In case of non-grid arrows then you can also have this as a unit which is recommended as it looks better. If the scalar is < 1 then the overlap is ignored.

box_prop If you want the boxes to have proportions indicating some other factors then input a matrix with quantiles for the proportions. Note the size mus be `nrow(transition_flow) x 2`.

mar A numerical vector of the form c(bottom, left, top, right) of the type `unit()`

main The title of the plot if any, default `NULL`

box_label A vector of length 2 if you want to label each box column

box_label_pos The position of the label, either `'top'` or `'bottom'`

box_label_cex The cex of the label, defaults to the default cex

color_bar If you have proportions inside the transition_flow variable then the color_bar will automatically appear at the bottom unless you set this to `FALSE`

color_bar_cex The size of the tick labels for the color bar

color_bar_labels
The labels of the two proportions that make up the color bar

color_bar_subspace
If there is little or no difference exists at the low/high proportions of the spectrum then it can be of interest to focus the color change to the center leaving the tails constant

new_page If you want the plot to appear on a new blank page then set this to `TRUE`, by default it is `FALSE`.

Value

void

Examples

```
## Not run:
# This example does not run since it
# takes a little while to assemble the
```

```
# arrows and RMD Check complains that this
# is more than allowed for
par_org <- par(ask=TRUE)
# Settings
no_boxes <- 3
# Generate test setting
transition_matrix <- matrix(NA, nrow=no_boxes, ncol=no_boxes)
transition_matrix[1,] <- 200*c(.5, .25, .25)
transition_matrix[2,] <- 540*c(.75, .10, .15)
transition_matrix[3,] <- 340*c(0, .2, .80)

grid.newpage()
transitionPlot(transition_matrix,
               box_txt = c("First", "Second", "Third"),
               type_of_arrow = "simple",
               min_lwd = unit(1, "mm"),
               max_lwd = unit(6, "mm"),
               overlap_add_width = unit(1, "mm"))


# Setup proportions
box_prop <- cbind(c(1,0,0.5), c(.52,.2,.8))
# From the Set2 Colorbrewer
start_box_clr <- c("#8DA0CB", "#FC8D62")
# Darken the colors slightly
end_box_clr <- c(colorRampPalette(c(start_box_clr[1], "#000000"))(10)[2],
                 colorRampPalette(c(start_box_clr[2], "#000000"))(10)[2])
# Create a new grid
grid.newpage()
transitionPlot(transition_matrix, box_prop=box_prop,
               fill_start_box=start_box_clr, fill_end_box=end_box_clr,
            txt_start_clr = c("#FFFFFF", "#000000"), txt_end_clr = c("#FFFFFF", "#000000"),
               box_txt = c("First", "Second", "Third"),
               type_of_arrow = "gradient",
               min_lwd = unit(1, "mm"),
               max_lwd = unit(10, "mm"),
               overlap_add_width = unit(1, "mm"))
par(par_org)

## End(Not run)
```

Index