Package 'forestplot'

June 12, 2025

Version 3.1.7

Title Advanced Forest Plot Using 'grid' Graphics

Description Allows the creation of forest plots with advanced features, such as multiple confidence intervals per row, customizable fonts for individual text elements, and flexible confidence interval drawing. It also supports mixing text with mathematical expressions. The package extends the application of forest plots beyond traditional meta-analyses, offering a more general version of the original 'rmeta' package's forestplot() function. It relies heavily on the 'grid' package for rendering the plots.

License GPL-2

URL https://gforge.se/packages/

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

Biarch yes

Depends R (>= 4.1.0), grid, checkmate, abind

Suggests dplyr, Gmisc, Greg, knitr, purrr, rlang, rmarkdown, rmeta, rms, testthat, tibble, tidyr, tidyselect, tidyverse

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

RoxygenNote 7.3.2

Author Max Gordon [aut, cre], Thomas Lumley [aut, ctb]

Maintainer Max Gordon <max@gforge.se>

Repository CRAN

Date/Publication 2025-06-12 09:40:02 UTC

Contents

`forestplot-package` *Package description*

Description

The forest plot function, `forestplot()`, 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.

Details

The forestplot:

1. Allows for multiple confidence intervals per row
2. Custom fonts for each text element
3. Custom confidence intervals
4. Text mixed with expressions
5. Legends both on top/left of the plot and within the graph
6. Custom line height including auto-adapt height
7. Graph width that auto-adapts
8. Flexible arguments
9. and more

Additional functions

The `getTicks` tries to format ticks for plots in a nicer way. The major use is for exponential form where ticks are generated using the 2^n since a doubling is a concept easy to grasp for less mathematical-savvy readers.

Author(s)

Maintainer: Max Gordon <max@gforge.se>

Authors:

- Thomas Lumley [contributor]

See Also

Useful links:

- https://gforge.se/packages/
- Report bugs at https://github.com/gforge/forestplot/issues

---

assertAndRetrieveTidyValue

Retriever of `tidyselect`

Description

As forestpot has evolved we now primarily use `tidyverse` select style. This function helps with backward compatibility

Usage

```
assertAndRetrieveTidyValue(
  x,
  value,
  name = deparse(substitute(value)),
  optional = FALSE
)
```

Arguments

`x`	The data with the potential value
`value`	The value
`name`	The name of the value
`optional`	Is the value optional

Value

value with attribute

dfHRQoL *Regression coefficients and confidence intervals from HRQoL study*

Description

The data is a dataframe with the Swedish and the Danish coefficients for health related quality of life (HRQoL) 1 year after total hip arthroplasty surgery. The age is modeled as a spline and is therefore presented as a contrast.

Author(s)

Max Gordon <max@gforge.se>

forestplot *Draws a forest plot*

Description

This function generates a forest plot with extended capabilities compared to the default `forestplot()` function in the `rmeta` package. It overcomes some limitations of the original function, including the addition of expressions, use of multiple confidence bands per label, autosizing to viewport, and uses modern tidyverse syntax. Refer to `vignette("forestplot")` for comprehensive details.

Usage

```
forestplot(...)

## S3 method for class 'data.frame'
forestplot(x, mean, lower, upper, labeltext, is.summary, boxsize, ...)

## Default S3 method:
forestplot(
  labeltext,
  mean,
  lower,
  upper,
  align = NULL,
  is.summary = FALSE,
  graph.pos = "right",
  hrzl_lines = NULL,
  clip = c(-Inf, Inf),
  xlab = NULL,
  zero = ifelse(xlog, 1, 0),
  graphwidth = "auto",
  colgap = NULL,
```

```
  lineheight = "auto",
  line.margin = NULL,
  col = fpColors(),
  txt_gp = fpTxtGp(),
  xlog = FALSE,
  xticks = NULL,
  xticks.digits = 2,
  grid = FALSE,
  lwd.xaxis = NULL,
  lwd.zero = 1,
  lwd.ci = NULL,
  lty.ci = 1,
  ci.vertices = NULL,
  ci.vertices.height = 0.1,
  boxsize = NULL,
  mar = unit(rep(5, times = 4), "mm"),
  title = NULL,
  legend = NULL,
  legend_args = fpLegend(),
  new_page = getOption("forestplot_new_page", TRUE),
  fn.ci_norm = fpDrawNormalCI,
  fn.ci_sum = fpDrawSummaryCI,
  fn.legend = NULL,
  shapes_gp = fpShapesGp(),
  ...
)

## S3 method for class 'gforge_forestplot'
print(x, ...)

## S3 method for class 'gforge_forestplot'
plot(x, y, ..., new_page = FALSE)

## S3 method for class 'grouped_df'
forestplot(x, labeltext, mean, lower, upper, legend, is.summary, boxsize, ...)
```

Arguments

`...`	Passed on to the `fn.ci_norm` and `fn.ci_sum` arguments
`x`	The `gforge_forestplot` object to be printed
`mean`	The name of the column if using the *dplyr* select syntax - defaults to "mean", else it should be a vector or a matrix with the averages. You can also provide a 2D/3D matrix that is automatically converted to the lower/upper parameters. The values should be in exponentiated form if they follow this interpretation, e.g. use exp(mean) if you have the output from a logistic regression
`lower`	The lower bound of the confidence interval for the forestplot, needs to be the same format as the mean.

upper — The upper bound of the confidence interval for the forestplot, needs to be the same format as the mean.

labeltext — A list, matrix, vector or expression with the names of each row or the name of the column if using the *dplyr* select syntax - defaults to "labeltext". Note that when using group_by a separate labeltext is not allowed. The list should be wrapped in m x n number to resemble a matrix: `list(list("rowname 1 col 1", "rowname 2 col 1"), list("r1c2", expression(beta))`. 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. *Note:* If you do not provide the mean/lower/upper arguments the function expects the label text to be a matrix containing the labeltext in the rownames and then columns for mean, lower, and upper.

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

boxsize — Override the default box size based on precision

align — Vector giving alignment (l,r,c) for the table columns

graph.pos — The position of the graph element within the table of text. The position can be `1-(ncol(labeltext) + 1)`. You can also choose set the position to `"left"` or `"right"`.

hrzl_lines — Add horizontal lines to graph. Can either be TRUE or a `list` of `gpar`. See line section below for details.

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

xlab — x-axis label

zero — x-axis coordinate for zero line. If you provide a vector of length 2 it will print a rectangle instead of just a line. If you provide NA the line is suppressed.

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

colgap — Sets the gap between columns, defaults to 6 mm but for relative widths. Note that the value should be in `unit(,"npc")`.

lineheight — Height of the graph. By default this is `auto` and adjusts to the space that is left after adjusting for x-axis size and legend. Sometimes it might be desirable 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% more than what the text height is as your line height

line.margin — Set the margin between rows, provided in numeric or `unit` form. When having multiple confidence lines per row setting the correct margin in order to visually separate rows

col — Set the colors for all the elements. See `fpColors` for details

txt_gp — Set the fonts etc for all text elements. See `fpTxtGp` for details

xlog — If TRUE, x-axis tick marks are to follow a logarithmic scale, e.g. for logistic regression (OR), survival estimates (HR), Poisson regression etc. *Note:* This is an intentional break with the original `forestplot` function as I've found that exponentiated ticks/clips/zero effect are more difficult to for non-statisticians and there are sometimes issues with rounding the tick marks properly.

xticks — Optional user-specified x-axis tick marks. Specify NULL to use the defaults, numeric(0) to omit the x-axis. By adding a labels-attribute, `attr(my_ticks, "labels") <- ...` you can dictate the outputted text at each tick. If you specify a boolean vector then ticks indicated with FALSE wont be printed. Note that the labels have to be the same length as the main variable.

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

grid — If you want a discrete gray dashed grid at the level of the ticks you can set this parameter to `TRUE`. If you set the parameter to a vector of values lines will be drawn at the corresponding positions. If you want to specify the `gpar` of the lines then either directly pass a `gpar` object or set the gp attribute e.g. `attr(line_vector, "gp") <- gpar(lty = 2, col = "red")`

lwd.xaxis — lwd for the xaxis, see `gpar`

lwd.zero — lwd for the vertical line that gives the no-effect line, see `gpar`

lwd.ci — lwd for the confidence bands, see `gpar`

lty.ci — lty for the confidence bands, see `gpar`

ci.vertices — Set this to TRUE if you want the ends of the confidence intervals to be shaped as a T. This is set default to TRUE if you have any other line type than 1 since there is a risk of a dash occurring at the very end, i.e. showing incorrectly narrow confidence interval.

ci.vertices.height — The height hoft the vertices. Defaults to npc units corresponding to 10% of the row height. *Note that the arrows correspond to the vertices heights.*

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

title — The title of the plot if any

legend — Legend corresponding to the number of bars

legend_args — The legend arguments as returned by the `fpLegend` function.

new_page — If you want the plot to appear on a new blank page then set this to `TRUE`, by default it is `TRUE`. If you want to change this behavior for all plots then set the `options(forestplot_new_page = FALSE)`

fn.ci_norm — 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`

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

fn.legend — 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

shapes_gp	Sets graphical parameters (squares and lines widths, styles, etc.) of all shapes drawn (squares, lines, diamonds, etc.). This overrides `col`, `lwd.xaxis`, `lwd.zero`, `lwd.ci` and `lty.ci`.
y	Ignored

Details

This version of `forestplot()` enhances the standard function in the following ways:

- **Adding Expressions:** Allows the use of expressions, such as `expression(beta)`.
- **Multiple Bands:** Enables multiple confidence bands for the same label.
- **Autosize:** Adapts to the viewport (graph) size.
- **Tidyverse syntax:** Utilizes convenient dplyr/tidyverse syntax for more flexible data manipulation.

Value

`gforge_forestplot` object

Multiple bands

Multiple bands (or lines) per variable can be useful for comparing different outcomes. For instance, you may want to compare heart disease-specific survival to overall survival rates for smokers. It can be insightful to overlay two bands for this purpose. Another application could be displaying crude and adjusted estimates as separate bands.

Horizontal lines

The `hrzl_lines` argument can be set as `TRUE` or a `list` with `grid::gpar` elements.

- `TRUE`: A line will be added based upon the `is.summary` rows. If the first line is a summary it
- `grid::gpar`: The same as above but the lines will be formatted according to the `grid::gpar` element
- `list`: The list must either be numbered, i.e. `list("2" = gpar(lty = 1))`, or have the same length as the `NROW(mean) + 1`. If the list is numbered the numbers should not exceed the `NROW(mean) + 1`. The no. *1 row designates the top*, i.e. the line above the first row, all other correspond to *the row below*. Each element in the list needs to be `TRUE`, `NULL`, or `gpar` element. The `TRUE` defaults to a standard line, the `NULL` skips a line, while `gpar` corresponds to the fully customized line. Apart from allowing standard `gpar` line descriptions, `lty`, `lwd`, `col`, and more you can also specify `gpar(columns = c(1:3, 5))` if you for instance want the line to skip a column.

Known Issues

- The x-axis does not completely adhere to the margin.
- Autosizing boxes may not always yield the best visual result; manual adjustment is recommended where possible.

API Changes from `rmeta` package's `forestplot`

- **xlog:** Outputs the axis in log() format, but the input data should be in antilog/exp format.
- **col:** The corresponding function in this package is `fpColors`.

Author(s)

Max Gordon, Thomas Lumley

See Also

`vignette("forestplot")`

Other forestplot functions: `fpColors()`, `fpDrawNormalCI()`, `fpLegend()`, `fpShapesGp()`, `fp_add_lines()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`, `fp_set_zebra_style()`

Examples

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

ask <- par(ask = TRUE)

# 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)
)
test_data |>
  forestplot(labeltext = row_names,
             mean = coef,
             lower = low,
             upper = high,
             zero = 1,
             cex  = 2,
             lineheight = "auto",
             xlab = "Lab axis txt") |>
  fp_add_header("Group") |>
  fp_set_style(lines = gpar(col = "darkblue"))

# Print two plots side by side using the grid
# package's layout option for viewports
fp1 <- test_data |>
  forestplot(labeltext = row_names,
             mean = coef,
             lower = low,
             upper = high,
             zero = 1,
             cex  = 2,
             lineheight = "auto",
```

```
            title = "Plot 1",
            xlab = "Lab axis txt")
fp2 <- test_data |>
  forestplot(labeltext = row_names,
            mean = coef,
            lower = low,
            upper = high,
            zero = 1,
            cex  = 2,
            lineheight = "auto",
            xlab = "Lab axis txt",
            title = "Plot 2",
            new_page = FALSE)

grid.newpage()
pushViewport(viewport(layout = grid.layout(1, 2)))
pushViewport(viewport(layout.pos.col = 1))
plot(fp1)
popViewport()
pushViewport(viewport(layout.pos.col = 2))
plot(fp2)
popViewport(2)

# An advanced example
library(dplyr)
library(tidyr)
test_data <- data.frame(id = 1:4,
                        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))

# Convert into dplyr formatted data
out_data <- test_data |>
  pivot_longer(cols = everything() & -id) |>
  mutate(group = gsub("(.+)([12])$", "\\2", name),
         name = gsub("(.+)([12])$", "\\1", name)) |>
  pivot_wider() |>
  group_by(id) |>
  mutate(col1 = lapply(id, \(x) ifelse(x < 4,
                                       paste("Category", id),
                                       expression(Category >= 4))),
         col2 = lapply(1:n(), \(i) substitute(expression(bar(x) == val),
                                              list(val = mean(coef) |> round(2)))),
         col2 = if_else(id == 1,
                        rep("ref", n()) |> as.list(),
                        col2)) |>
  group_by(group)

out_data |>
  forestplot(mean = coef,
```

```
              lower = low,
              upper = high,
              labeltext = c(col1, col2),
              title = "Cool study",
              zero = c(0.98, 1.02),
              grid = structure(c(2^-.5, 2^.5),
                               gp = gpar(col = "steelblue", lty = 2)
              ),
              boxsize = 0.25,
              xlab = "The estimates",
              new_page = TRUE,
              legend = c("Treatment", "Placebo"),
              legend_args = fpLegend(
                pos = list("topright"),
                title = "Group",
                r = unit(.1, "snpc"),
                gp = gpar(col = "#CCCCCC", lwd = 1.5)
              )) |>
  fp_set_style(box = c("royalblue", "gold"),
               line = c("darkblue", "orange"),
               summary = c("darkblue", "red"))

# An example of how the exponential works
data.frame(coef = c(2.45, 0.43),
           low = c(1.5, 0.25),
           high = c(4, 0.75),
           boxsize = c(0.25, 0.25),
           variables = c("Variable A", "Variable B")) |>
  forestplot(labeltext = c(variables, coef),
             mean = coef,
             lower = low,
             upper = high,
             boxsize = boxsize,
             zero = 1,
             xlog = TRUE) |>
  fp_set_style(lines = "red", box = "darkred") |>
  fp_add_header(coef = "HR" |> fp_txt_plain() |> fp_align_center(),
                variables = "Measurements")

# An example using style
forestplot(labeltext = cbind(Author = c("Smith et al", "Smooth et al", "Al et al")),
           mean = cbind(1:3, 1.5:3.5),
           lower = cbind(0:2, 0.5:2.5),
           upper = cbind(4:6, 5.5:7.5),
           is.summary = c(FALSE, FALSE, TRUE),
           vertices = TRUE) |>
 fp_set_style(default = gpar(lineend = "square", linejoin = "mitre", lwd = 3, col = "pink"),
              box = gpar(fill = "black", col = "red"), # only one parameter
              lines = list( # as many parameters as CI
                gpar(lwd = 10), gpar(lwd = 5),
                gpar(), gpar(),
                gpar(lwd = 2), gpar(lwd = 1)
              ),
```

```
                summary = list( # as many parameters as band per label
                  gpar(fill = "violet", col = "gray", lwd = 10),
                  gpar(fill = "orange", col = "gray", lwd = 10)
                ))

par(ask = ask)
# See vignette for a more detailed description
# vignette("forestplot",  package="forestplot")
```

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

Description

This function encapsulates all the colors that are used in the `forestplot` 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",
  hrz_lines = "black",
  vrtcl_lines = "lightgray"
)
```

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
`hrz_lines`	The color of the horizontal lines
`vrtcl_lines`	The color of the vertical lines

Details

Further customization of non-text elements can be performed with fpShapesGp passed as shapes_gp parameter to forestplot. The fpColors function is kept for backwards compatibility.

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

A list with key elements

Author(s)

Max Gordon, Thomas Lumley

See Also

Other forestplot functions: forestplot(), fpDrawNormalCI(), fpLegend(), fpShapesGp(), fp_add_lines(), fp_decorate_graph(), fp_insert_row(), fp_set_style(), fp_set_zebra_style()

Examples

```
ask <- par(ask = TRUE)

# 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)

forestplot(
  labeltext = row_names,
  test_data[, c("coef", "low", "high")],
  is.summary = c(TRUE, FALSE, FALSE),
  boxsize = test_data$boxsize,
  zero = 1,
  xlog = TRUE,
  col = fpColors(lines = "#990000", box = "#660000", zero = "darkblue"),
  new_page = TRUE
)

par(ask = ask)
```

`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,
  lty = 1,
  vertices,
  vertices.height = 0.1,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)

fpDrawDiamondCI(
  lower_limit,
  estimate,
  upper_limit,
  size,
  y.offset = 0.5,
  clr.line,
  clr.marker,
  lwd,
  lty = 1,
  vertices,
  vertices.height = 0.1,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)

fpDrawCircleCI(
  lower_limit,
  estimate,
```

```
  upper_limit,
  size,
  y.offset = 0.5,
  clr.line,
  clr.marker,
  lwd,
  lty = 1,
  vertices,
  vertices.height = 0.1,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)

fpDrawPointCI(
  lower_limit,
  estimate,
  upper_limit,
  size,
  y.offset = 0.5,
  clr.line,
  clr.marker,
  lwd,
  lty = 1,
  vertices,
  vertices.height = 0.1,
  pch = 1,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)

fpDrawSummaryCI(
  lower_limit,
  estimate,
  upper_limit,
  size,
  col,
  y.offset = 0.5,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)

fpDrawBarCI(
  lower_limit,
  estimate,
  upper_limit,
```

```
  size,
  col,
  y.offset = 0.5,
  shapes_gp = fpShapesGp(),
  shape_coordinates = structure(c(1, 1), max.coords = c(1, 1)),
  ...
)
```

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, see gpar
lty	Line type, see gpar
vertices	Set this to TRUE if you want the ends of the confidence intervals to be shaped as a T. This is set default to TRUE if you have any other line type than 1 since there is a risk of a dash occurring at the very end, i.e. showing incorrectly narrow confidence interval.
vertices.height	The height hoft the vertices. Defaults to npc units corresponding to 10% of the row height.
shapes_gp	A set of graphical parameters of class fpShapesGp
shape_coordinates	A vector of length 2 the label (first item of the vector) and the band (second item of the vector) of the confidence interval. This is used together with shapes_gp to retrieve graphical parameters for that item.
...	Allows additional parameters for sibling functions
pch	Type of point see grid.points for details
col	The color of the summary object

Value

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

Author(s)

Max Gordon, Thomas Lumley

See Also

Other forestplot functions: `forestplot()`, `fpColors()`, `fpLegend()`, `fpShapesGp()`, `fp_add_lines()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`, `fp_set_zebra_style()`

Examples

```
ask <- par(ask = TRUE)

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
```

```
forestplot(row_names, coef, low, high,
  fn.ci_norm = fpDrawDiamondCI,
  title = "Cool study",
  zero = 1, boxsize = 0.25,
  col = fpColors(
    box = c("royalblue", "gold"),
    line = c("darkblue", "orange"),
    summary = c("darkblue", "red")
  ),
  xlab = "The estimates",
  new_page = TRUE,
  legend = c("Treatment", "Placebo"),
  legend_args = fpLegend(
    title = "Group",
    pos = list("topright", inset = .1),
    r = unit(.1, "snpc"),
    gp = gpar(col = "#CCCCCC", lwd = 1.5)
  )
)

# Change first to diamonds
forestplot(row_names, coef, low, high,
  fn.ci_norm = c(
    "fpDrawDiamondCI",
    rep("fpDrawNormalCI",
      times = nrow(coef) - 1
    )
  ),
  title = "Cool study",
  zero = 1, boxsize = 0.25,
  col = fpColors(
    box = c("royalblue", "gold"),
    line = c("darkblue", "orange"),
    summary = c("darkblue", "red")
  ),
  xlab = "The estimates",
  new_page = TRUE,
  legend = c("Treatment", "Placebo"),
  legend_args = fpLegend(
    title = "Group",
    pos = list("topright", inset = .1),
    r = unit(.1, "snpc"),
    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]]
forestplot(row_names, coef, low, high,
  fn.ci_norm = list(
```

```
    list(fpDrawDiamondCI, fpDrawCircleCI),
    list(fpDrawNormalCI, fpDrawNormalCI),
    list(fpDrawNormalCI, fpDrawCircleCI),
    list(fpDrawNormalCI, fpDrawNormalCI)
  ),
  title = "Cool study",
  zero = 1, boxsize = 0.25,
  col = fpColors(
    box = c("royalblue", "gold"),
    line = c("darkblue", "orange"),
    summary = c("darkblue", "red")
  ),
  xlab = "The estimates",
  new_page = TRUE,
  legend = c("Treatment", "Placebo"),
  legend_args = fpLegend(
    title = "Group",
    pos = list("topright", inset = .1),
    r = unit(.1, "snpc"),
    gp = gpar(col = "#CCCCCC", lwd = 1.5)
  )
)

par(ask = ask)
```

`fpLegend` *A function for the legend used in forestplot()*

Description

This function encapsulates all the legend options that are used in the `forestplot` function. This is in order to limit the crowding among the arguments for the `forestplot` call.

Usage

```
fpLegend(
  pos = "top",
  gp = NULL,
  r = unit(0, "snpc"),
  padding = unit(ifelse(!is.null(gp), 3, 0), "mm"),
  title = NULL
)
```

Arguments

`pos`	The position of the legend, either at the "top" or the "right" unless 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 `pos`

= list("topright") - this is equivalent to 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. 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. pos = list("topright", "inset" = .1, "align" = "horizontal")

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

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.

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.

title — The title of the legend if any

Value

list Returns a list with all the elements

See Also

Other forestplot functions: forestplot(), fpColors(), fpDrawNormalCI(), fpShapesGp(), fp_add_lines(), fp_decorate_graph(), fp_insert_row(), fp_set_style(), fp_set_zebra_style()

---

fpShapesGp *A function for graphical parameters of the shapes used in forestplot()*

Description

This function encapsulates all the non-text elements that are used in the forestplot() function. As there are plenty of shapes options this function gathers them all in one place.

Usage

```
fpShapesGp(
  default = NULL,
  box = NULL,
  lines = NULL,
  vertices = NULL,
  summary = NULL,
  zero = NULL,
  axes = NULL,
```

```
  hrz_lines = NULL,
  vrtcl_lines = NULL,
  grid = NULL
)
```

Arguments

default	A fallback grid::gpar for all unspecified attributes. If set to NULL then it defaults to legacy parameters, including the col, lwd.xaxis, lwd.ci and lty.ci parameter of fpColors.
box	The graphical parameters (gpar, character) of the box, circle or point indicating the point estimate, i.e. the middle of the confidence interval (may be a list of gpars). If provided a string a gpar will be generated with col, and fill for those arguments.
lines	The graphical parameters (gpar, character) of the confidence lines (may be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
vertices	The graphical parameters (gpar, character) of the vertices (may be a list of gpars). If ci.vertices is set to TRUE in forestplot vertices inherits from lines all its parameters but lty that is set to "solid" by default.
summary	The graphical parameters (gpar, character) of the summary (may be a list of gpars). If provided a string a gpar will be generated with col, and fill for those arguments.
zero	The graphical parameters (gpar) of the zero line (may not be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
axes	The graphical parameters (gpar) of the x-axis at the bottom (may not be a list of gpars).
hrz_lines	The graphical parameters (gpar) of the horizontal lines (may not be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
vrtcl_lines	The graphical parameters (gpar) of the vertical lines (may not be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
grid	The graphical parameters (gpar) of the grid (vertical lines) (may be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.

Details

This function obsoletes fpColors().

If some, but not all parameters of a shape (e.g. box) are specified in gpar() such as setting lwd but not line color, the unspecified parameters default to the ones specified in default, then, default to legacy parameters of forestplot such as col.

Parameters box, lines, vertices, summary may be set as list containing several gpars. The length of the list must either be equal to the number of bands per label or to the number of bands multiplied by the number of labels, allowing specification of different styles for different parts of the forest plot.

The parameter `grid` can either be a single gpar or a list of gpars with as many elements as there are lines in the grid (as set by the `xticks` or `grid` arguments of forestplot)

Parameters `zero`, `axes`, `hrz_lines` must either be NULL or gpar but cannot be lists of gpars.

Value

list A list with the elements:

- default: the gpar for default attributes
- box: the gpar or list of gpars of the box/marker
- lines: the gpar or list of gpars of the lines
- vertices: the gpar or list of gpars of the vertices
- summary: the gpar or list of gpars of the summary
- zero: the gpar of the zero vertical line
- axes: the gpar of the x-axis
- hrz_lines: the gpar of the horizontal lines
- grid: the gpar or list of gpars of the grid lines

Author(s)

Andre GILLIBERT

See Also

Other forestplot functions: `forestplot()`, `fpColors()`, `fpDrawNormalCI()`, `fpLegend()`, `fp_add_lines()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`, `fp_set_zebra_style()`

Examples

```
ask <- par(ask = TRUE)

# An example of how fpShapesGp works

styles <- fpShapesGp(
  default = gpar(col = "pink", lwd = 2, lineend = "square", linejoin = "mitre"),
  grid = list(
    gpar(col = "blue"),
    gpar(col = "black"),
    gpar(col = "blue")
  ),
  box = list(
    gpar(fill = "black"),
    gpar(fill = "blue"),
    gpar(fill = "black"),
    gpar(fill = "blue")
  ),
  lines = gpar(lty = "dashed"),
  vertices = gpar(lwd = 5, col = "red")
)
```

```
forestplot(
  labeltext = c("Author1", "Author2", "Author3", "Author4"),
  grid = c(1, 3, 5),
  mean = 1:4, lower = 0:3, upper = 2:5,
  shapes_gp = styles
)

par(ask = ask)
```

`fpTxtGp` *Get font settings for forestplot*

Description

This function generates all the `gpar()` elements for the different text elements within the graph. Elements not specified inherit their default settings from the `label` argument.

Usage

```
fpTxtGp(label, summary, xlab, title, ticks, legend, legend.title, cex = 1)
```

Arguments

`label`	The text labels (see details below)
`summary`	The summary labels (see details below)
`xlab`	The xlab text
`title`	The plot title
`ticks`	The ticks associated with the xlab
`legend`	The legend text
`legend.title`	The legend title
`cex`	The font size

Value

A list of the `fpTxtGp` class

List arguments for `label`/`summary`

You can provide a `list` of elements for the `label` and `summary` in order to specify separate elements. If you provide a `list` in one dimension the `gpar` elements are assumed to follow the columns. If you provide a `list` of 2 dimensions the structure assumes is `list[[row]][[column]]` and the number of elements should correspond to the number of labels for the `label` argument, i.e. without the rows marked as summary elements. The same goes for `summary` arguments.

Examples

```
fpTxtGp(label = gpar(fontfamily = "HersheySerif"))
```

`fp_add_lines` *Adds a line to the graph*

Description

Adds a line to the graph, defaults to horizontal. Lines with prefix `v_` will be vertical, no prefix or `h_` will be horizontal. If argument is `TRUE` or just empty: A line will be added based upon the `is.summary` rows. If the first line is a summary it will choose the last non-summary row.

Usage

```
fp_add_lines(x, ...)
```

Arguments

`x`	The forestplot object
`...`	A set of arguments. Can either be `TRUE` or a set of numbered arguments with `gpar`. See line section below for details.

Details

If you provide the argument as a number it will add the line to that particular line. 1 corresponds to the top row and the max row is `num_rows + 1`. If the argument is `TRUE` it will default to a standard line. A string will default to the color of that string. If you provide a grid::gpar element it will style the line according to the `gpar object`. Apart from allowing standard `gpar` line descriptions, `lty`, `lwd`, `col`, and more you can also specify `gpar(columns = c(1:3, 5))` if you for instance want the line to skip a column.

If you want to add mix vertical and horizontal lines you can prefix the lines with `h_` and `v_`, e.g. `v_2` for the second column.

Value

The foresplot object with the styles

See Also

Other graph modifiers: `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`, `fp_set_zebra_style()`

Other forestplot functions: `forestplot()`, `fpColors()`, `fpDrawNormalCI()`, `fpLegend()`, `fpShapesGp()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`, `fp_set_zebra_style()`

Examples

```
base_data <- tibble::tibble(mean  = c(0.578, 0.165, 0.246, 0.700, 0.348, 0.139, 1.017),
                            lower = c(0.372, 0.018, 0.072, 0.333, 0.083, 0.016, 0.365),
                            upper = c(0.898, 1.517, 0.833, 1.474, 1.455, 1.209, 2.831),
                            study = c("Auckland", "Block", "Doran", "Gamsu",
                                      "Morrison", "Papageorgiou", "Tauesch"),
```

```
                            deaths_steroid = c("36", "1", "4", "14", "3", "1", "8"),
                            deaths_placebo = c("60", "5", "11", "20", "7", "7", "10"),
                          OR = c("0.58", "0.16", "0.25", "0.70", "0.35", "0.14", "1.02"))

base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
  fp_set_style(box = "royalblue",
               line = "darkblue") |>
  fp_add_lines("steelblue")


base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
  fp_set_style(box = "royalblue",
               line = "darkblue") |>
  # Add top line
  fp_add_lines(h_3 = "darkred") |>
  # Add surrounding box with fancy syntax
  fp_add_lines(h_5 = gpar(col = "steelblue", columns = 1:4, lty = 2),
               h_7 = gpar(col = "steelblue", columns = 1:4, lty = 2),
               v_1 = gpar(col = "steelblue", rows = 5:6, lty = 3, lty = 2),
               v_5 = gpar(col = "steelblue", rows = 5:6, lty = 3, lty = 2))
```

`fp_decorate_graph` *Decorate the graph*

Description

Decorate the graph

Usage

```
fp_decorate_graph(
  x,
  box = NULL,
  right_bottom_txt = NULL,
  left_bottom_txt = NULL,
  right_top_txt = NULL,
```

```
  left_top_txt = NULL,
  grid = NULL,
  graph.pos = NULL
)
```

Arguments

x	The forestplot object
box	Decorate the graph by framing it in a box. If provided TRUE it will simply frame the graph in a black box. If you provide a string it is assumed to be the color of the graph. Acceptable arguments are also gpar() and a grob object to draw.
right_bottom_txt	Text to appear at the right bottom of the graph. Can be decorated fp_txt_* functions.
left_bottom_txt	Text to appear at the left bottom of the graph. Can be decorated fp_txt_* functions.
right_top_txt	Text to appear at the right top of the graph. Can be decorated fp_txt_* functions.
left_top_txt	Text to appear at the left top of the graph. Can be decorated fp_txt_* functions.
grid	If you want a discrete gray dashed grid at the level of the ticks you can set this parameter to TRUE. If you set the parameter to a vector of values lines will be drawn at the corresponding positions. If you want to specify the gpar of the lines then either directly pass a gpar object or set the gp attribute e.g. attr(line_vector, "gp") <- gpar(lty = 2, col = "red")
graph.pos	The position of the graph element within the table of text. The position can be 1-(ncol(labeltext) + 1). You can also choose set the position to "left" or "right".

Value

The forestplot object with the extended decoration

See Also

Other graph modifiers: fp_add_lines(), fp_insert_row(), fp_set_style(), fp_set_zebra_style()

Other forestplot functions: forestplot(), fpColors(), fpDrawNormalCI(), fpLegend(), fpShapesGp(), fp_add_lines(), fp_insert_row(), fp_set_style(), fp_set_zebra_style()

Examples

```
base_data <- tibble::tibble(mean  = c(0.578, 0.165, 0.246, 0.700, 0.348, 0.139, 1.017),
                            lower = c(0.372, 0.018, 0.072, 0.333, 0.083, 0.016, 0.365),
                            upper = c(0.898, 1.517, 0.833, 1.474, 1.455, 1.209, 2.831),
                            study = c("Auckland", "Block", "Doran", "Gamsu",
                                      "Morrison", "Papageorgiou", "Tauesch"),
                            deaths_steroid = c("36", "1", "4", "14", "3", "1", "8"),
                            deaths_placebo = c("60", "5", "11", "20", "7", "7", "10"),
                           OR = c("0.58", "0.16", "0.25", "0.70", "0.35", "0.14", "1.02"))
```

```
base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
  fp_set_style(box = "royalblue",
               line = "darkblue",
               summary = gpar(fill = "royalblue", clr = "black"),
               txt_gp = fpTxtGp(label = gpar(fontfamily = "mono"))) |>
  fp_decorate_graph(box = "lightgray",
                    right_bottom_txt = fp_txt_gp("RB", gp = gpar(cex = .5)),
                    left_bottom_txt = fp_txt_gp("LB", gp = gpar(cex = .5)),
                    right_top_txt = "RT",
                    left_top_txt = "LT")
```

`fp_insert_row` *Insert/append rows into forestplot*

Description

These functions are used for inserting or appending a row into a forestplot object. Can be used for inputting multiple rows. Just make sure that all elements are of equal length.

Usage

```
fp_insert_row(
  x,
  ...,
  mean = NULL,
  lower = NULL,
  upper = NULL,
  position = 1,
  is.summary = FALSE,
  boxsize = NA
)

fp_add_header(x, ..., position = 1, is.summary = TRUE)

fp_append_row(x, ..., position = "last", is.summary = FALSE)
```

Arguments

`x`	The forestplot object
`...`	Either named arguments that correspond to the original column names or unnamed arguments that will map in appearing order.

mean	Either a mean or all the values if three columns (mean, lower, upper)
lower	A vector or matrix with the lower confidence interval
upper	A vector or matrix with the upper confidence interval
position	The row position to input at. Either a row number or "last".
is.summary	Whether the row is a summary.
boxsize	The box size for the drawn estimate line

Value

The foresplot object with the added rows

See Also

Other graph modifiers: fp_add_lines(), fp_decorate_graph(), fp_set_style(), fp_set_zebra_style()

Other forestplot functions: forestplot(), fpColors(), fpDrawNormalCI(), fpLegend(), fpShapesGp(), fp_add_lines(), fp_decorate_graph(), fp_set_style(), fp_set_zebra_style()

Examples

```
base_data <- tibble::tibble(mean  = c(0.578, 0.165, 0.246, 0.700, 0.348, 0.139, 1.017),
                            lower = c(0.372, 0.018, 0.072, 0.333, 0.083, 0.016, 0.365),
                            upper = c(0.898, 1.517, 0.833, 1.474, 1.455, 1.209, 2.831),
                            study = c("Auckland", "Block", "Doran", "Gamsu",
                                      "Morrison", "Papageorgiou", "Tauesch"),
                            deaths_steroid = c("36", "1", "4", "14", "3", "1", "8"),
                            deaths_placebo = c("60", "5", "11", "20", "7", "7", "10"),
                           OR = c("0.58", "0.16", "0.25", "0.70", "0.35", "0.14", "1.02"))

base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
  fp_append_row(mean  = 0.531,
                lower = 0.386,
                upper = 0.731,
                study = "Summary",
                OR = "0.53",
                is.summary = TRUE)
```

`fp_set_style` *Set the style of the graph*

Description

Sets the output style associated with the `foresplot`

Usage

```
fp_set_style(
  x,
  default = NULL,
  box = NULL,
  lines = NULL,
  vertices = NULL,
  summary = NULL,
  zero = NULL,
  axes = NULL,
  hrz_lines = NULL,
  grid = NULL,
  txt_gp = NULL,
  align = NULL
)
```

Arguments

`x`	The forestplot object
`default`	A fallback grid::gpar for all unspecified attributes. If set to NULL then it defaults to legacy parameters, including the `col`, `lwd.xaxis`, `lwd.ci` and `lty.ci` parameter of `fpColors`.
`box`	The graphical parameters (`gpar`, `character`) of the box, circle or point indicating the point estimate, i.e. the middle of the confidence interval (may be a list of gpars). If provided a string a `gpar` will be generated with `col`, and `fill` for those arguments.
`lines`	The graphical parameters (`gpar`, `character`) of the confidence lines (may be a list of gpars). If provided a string a `gpar` will be generated with `col` as the only arguments.
`vertices`	The graphical parameters (`gpar`, `character`) of the vertices (may be a list of gpars). If `ci.vertices` is set to TRUE in `forestplot` `vertices` inherits from `lines` all its parameters but lty that is set to "solid" by default.
`summary`	The graphical parameters (`gpar`, `character`) of the summary (may be a list of gpars). If provided a string a `gpar` will be generated with `col`, and `fill` for those arguments.
`zero`	The graphical parameters (`gpar`) of the zero line (may not be a list of gpars). If provided a string a `gpar` will be generated with `col` as the only arguments.

axes	The graphical parameters (gpar) of the x-axis at the bottom (may not be a list of gpars).
hrz_lines	The graphical parameters (gpar) of the horizontal lines (may not be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
grid	The graphical parameters (gpar) of the grid (vertical lines) (may be a list of gpars). If provided a string a gpar will be generated with col as the only arguments.
txt_gp	Set the fonts etc for all text elements. See fpTxtGp() for details
align	Vector giving alignment (l,r,c) for the table columns

Value

The foresplot object with the styles

See Also

Other graph modifiers: fp_add_lines(), fp_decorate_graph(), fp_insert_row(), fp_set_zebra_style()

Other forestplot functions: forestplot(), fpColors(), fpDrawNormalCI(), fpLegend(), fpShapesGp(), fp_add_lines(), fp_decorate_graph(), fp_insert_row(), fp_set_zebra_style()

Examples

```
base_data <- tibble::tibble(mean  = c(0.578, 0.165, 0.246, 0.700, 0.348, 0.139, 1.017),
                            lower = c(0.372, 0.018, 0.072, 0.333, 0.083, 0.016, 0.365),
                            upper = c(0.898, 1.517, 0.833, 1.474, 1.455, 1.209, 2.831),
                            study = c("Auckland", "Block", "Doran", "Gamsu",
                                      "Morrison", "Papageorgiou", "Tauesch"),
                            deaths_steroid = c("36", "1", "4", "14", "3", "1", "8"),
                            deaths_placebo = c("60", "5", "11", "20", "7", "7", "10"),
                           OR = c("0.58", "0.16", "0.25", "0.70", "0.35", "0.14", "1.02"))

base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
  fp_set_style(box = "royalblue",
               line = "darkblue",
               summary = gpar(fill = "royalblue", clr = "black"),
               txt_gp = fpTxtGp(label = gpar(fontfamily = "mono")))
```

`fp_set_zebra_style` *Decorate the plot with a zebra pattern*

Description

Decorate the plot with a zebra pattern

Usage

```
fp_set_zebra_style(x, ..., ignore_subheaders = FALSE)
```

Arguments

`x`	The forestplot object
`...`	The styles for each row
`ignore_subheaders`	The zebra will automatically restart at sub-headers, i.e. when there is a *summary* row that doesn't have any values.

Value

The forestplot object with the zebra style

See Also

Other graph modifiers: `fp_add_lines()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`

Other forestplot functions: `forestplot()`, `fpColors()`, `fpDrawNormalCI()`, `fpLegend()`, `fpShapesGp()`, `fp_add_lines()`, `fp_decorate_graph()`, `fp_insert_row()`, `fp_set_style()`

Examples

```
base_data <- tibble::tibble(mean  = c(0.578, 0.165, 0.246, 0.700, 0.348, 0.139, 1.017),
                            lower = c(0.372, 0.018, 0.072, 0.333, 0.083, 0.016, 0.365),
                            upper = c(0.898, 1.517, 0.833, 1.474, 1.455, 1.209, 2.831),
                            study = c("Auckland", "Block", "Doran", "Gamsu",
                                      "Morrison", "Papageorgiou", "Tauesch"),
                            deaths_steroid = c("36", "1", "4", "14", "3", "1", "8"),
                            deaths_placebo = c("60", "5", "11", "20", "7", "7", "10"),
                           OR = c("0.58", "0.16", "0.25", "0.70", "0.35", "0.14", "1.02"))

base_data |>
  forestplot(labeltext = c(study, deaths_steroid, deaths_placebo, OR),
             clip = c(0.1, 2.5),
             xlog = TRUE) |>
  fp_add_header(study = c("", "Study"),
                deaths_steroid = c("Deaths", "(steroid)"),
                deaths_placebo = c("Deaths", "(placebo)"),
                OR = c("", "OR")) |>
```

```
fp_set_style(box = "royalblue",
             line = "darkblue",
             summary = gpar(fill = "royalblue", clr = "black")) |>
fp_set_zebra_style("#EFEFEF")
```

`fp_txt_italic` *Text styling*

Description

This is a collection of functions to allow styling of text

Usage

```
fp_txt_italic(txt)

fp_txt_bold(txt)

fp_txt_plain(txt)

fp_txt_gp(txt, gp)

fp_align_left(txt)

fp_align_center(txt)

fp_align_right(txt)
```

Arguments

`txt`	The text to styl
`gp`	A `grid::gpar()` style to apply

Value

A list of txt with style attributes

Examples

```
fp_txt_italic("Italic text")
```

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

vector Returns a vector with the ticks

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
)
```

HRQoL *Regression coefficients and confidence intervals from HRQoL study*

Description

The data is a list containing the Swedish and the Danish coefficients for health related quality of life (HRQoL) 1 year after total hip arthroplasty surgery. The age is modeled as a spline and is therefore presented as a contrast.

Author(s)

Max Gordon <max@gforge.se>

---

prDefaultGp *Construct default parameters from arguments that may include missing arguments*

Description

Construct default parameters from arguments that may include missing arguments

Usage

```
prDefaultGp(col, lwd, lty)
```

Arguments

col	Line color (or missing)
lwd	Line width (or missing)
lty	Line type (or missing)

Value

a gpar object containing these three attributes

---

prepAlign *Prepares graph position*

Description

Prepares the graph position so that it matches the label size

Usage

```
prepAlign(align, graph.pos, nc)
```

Arguments

align	Vector giving alignment (l,r,c) for the table columns
graph.pos	An integer indicating the position of the graph
nc	The number of columns

Value

Returns vector of "l", "c", "r" values

---

prepGraphPositions *Prepares graph position*

Description

Prepares the graph position so that it matches the label size

Usage

```
prepGraphPositions(graph.pos, nc)
```

Arguments

graph.pos	The position of the graph element within the table of text. The position can be 1-(ncol(labeltext) + 1). You can also choose set the position to "left" or "right".
nc	The number of columns

Value

Returns number indicating the graph position

prepGridMargins *Convert margins to viewport npc margins*

Description

Convert margins to viewport npc margins

Usage

```
prepGridMargins(mar)
```

Arguments

mar	A vector of margins, at positions: • 1 = bottom • 2 = left • 3 = top • 4 = right

Value

Returns a list with `bottom`, `left`, `top`, and `right` as `unit("npc")`

prGetShapeGp *A function to extract graphical parameters from a fpShapesGp object*

Description

A function to extract graphical parameters from a fpShapesGp object

Usage

```
prGetShapeGp(
  shapes_gp,
  coords,
  object,
  default = grid::gpar(),
  nodefault = FALSE
)
```

Arguments

shapes_gp	An object of class fpShapesGp specifying all graphical parameters
coords	A numeric vector of length 2, specifying the label number (first item of the vector) and the confidence band number within this label ; that can be >= 2 if there are multiple confidence bands per label. Can be NULL for objects that are used only once (e.g. axes). Vector coords must have an R attribute max.coords as numeric vector of length 2 specifying the total number of labels and number of confidence bands by label for the forest plot. The first coordinate specify the label number and the second coordinate (for multi-band forest plots) specifies the band number within the label.
object	One of "box", "lines", "vertices", "summary", "zero", "axes", "hrz_lines" or "grid", refering to the object for which the graphical parameters are requested.
default	Default attributes to rely on when neither found in shapes_gp$object nor in shapes_gp$default
nodefault	Logical. If TRUE, do not search attribute in shapes_gp$default

Value

An object of class gpar

Author(s)

Andre GILLIBERT

---

prMergeGp *A function to merge two sets of graphical parameters*

Description

A function to merge two sets of graphical parameters

Usage

```
prMergeGp(weak = gpar(), strong = gpar())
```

Arguments

weak	A gpar
strong	Another gpar, with parameters taking precedence over weak

Value

A gpar merging attributes of both weak and strong

`safeLoadPackage` *Safely loads package*

Description

Stops if the package doesn't exist

Usage

```
safeLoadPackage(package)
```

Arguments

`package`	string naming the package/name space to load.

Index