Package ‘zenplots’

March 31, 2019

Version 0.0-4

Encoding UTF-8

Title Zigzag Expanded Navigation Plots

Description Graphical tools for visualizing high-dimensional data with a path of pairs. Note that this requires 'graph' from Bioconductor.
If you want to use dynamic graphics based on 'loon', you also need to have 'loon' installed, see <https://github.com/waddella/loon>.

Author Marius Hofert [aut, cre],
Wayne Oldford [aut]

Maintainer Marius Hofert <marius.hofert@uwaterloo.ca>

Depends R (>= 3.4.0)

Imports grid, graphics, stats, methods, MASS, graph, PairViz

Suggests loon, knitr, rmarkdown, Rgraphviz, ADGofTest, copula, Matrix, pcaPP, qqtest, qrmdata, qrmtools, rugarch, zoo, ggplot2, lattice, gridExtra, scagnostics

Enhances

License GPL-2 | GPL-3

NeedsCompilation yes

VignetteBuilder knitr, rmarkdown

Repository CRAN

Date/Publication 2019-03-31 16:30:03 UTC

R topics documented:

burst *Splitting an Input Object into a List of Columns*

Description

Split a (numeric/logical/character) vector, matrix, data.frame or a list of such into a list of columns, with corresponding group and variable information and labels.

Usage

```
burst(x, labs = list())
```

Arguments

x	numeric vector, matrix, data.frame or, for burst(), a list of such.
labs	either NULL (in which case neither group nor variable labels are computed) or a list containing the components group (either NULL, the group label basename or labels for the groups), var (either NULL, the variable label basename or labels for the variables), sep (the separator between group and variable labels) and group2d (a logical indicating whether labels of group_2d_*() plots are affected by group = NULL or still printed). If any of these components is not given, it is set to the defaults as can be found in zenplot(). Note that if at least one (group or variable) label is given in x, then those (original) labels will be used.

Value

burst() returns a list of length five, containing all columns of x (possibly with constructed group and variable names), the group and variable numbers (indices), and the group and variable labels.

Author(s)

Marius Hofert

Examples

```
## Unnamed list of (some named, some unnamed) valid components
A <- matrix(1:12, ncol = 3)
x <- list(A, 1:4, as.data.frame(A))
burst(x, labs = list(group = "G", var = "V", sep = ", "))
burst(x) # the same defaults as above
burst(x, labs = list(sep = " ")) # only changing the separator
## Note: - No group labels are given in 'x' and thus they are constructed
##         in the above call
##       - The variable names are only constructed if not given
burst(x, labs = list(group = ""))
burst(x, labs = list(group = NULL, group2d = TRUE)) # no group labels
## Note: There's no effect of 'group2d = TRUE' visible here as
##       'x' doesn't contain group labels
burst(x, labs = list(group = NULL)) # no group labels unless groups change
burst(x, labs = list(var = NULL)) # no variable labels
burst(x, labs = list(group = NULL, var = NULL)) # neither one
burst(x, labs = NULL) # similarly, without any labels at all

## Named list
x <- list(mat = A, vec = 1:4, df = as.data.frame(A))
burst(x)
## Note: - The given group labels are used
##       - The variable names are only constructed if not given
burst(x, labs = list(group = NULL, group2d = TRUE)) # no group labels
burst(x, labs = list(group = NULL)) # no group labels unless groups change
## Note: Now the effect of 'group2d' is visible.

## Partially named list
x <- list(mat = A, vec = 1:4, as.data.frame(A))
burst(x)
burst(x, labs = list(group = NULL, group2d = TRUE)) # no group labels
burst(x, labs = list(group = NULL)) # no group labels unless groups change
burst(x, labs = list(var = NULL)) # no variable labels
burst(x, labs = list(group = NULL, var = NULL)) # only group labels and only if groups change
burst(x, labs = NULL) # neither group nor variable labels
```

de_elect *German Election Data from 2002 and 2005*

Description

Data set consisting of 68 columns of data about the German elections 2002 and 2005.

Usage

```
data("de_elect")
```

Format

A `data.frame()` with 68 columns:

`District`: electoral district

`State`: federal state (Bundesland)

`Num.comm`: number of communities

`Area`: area 2004-12-31 (in square km)

`Pop`: population 2004-12-31 (in 1000)

`Men`: men (in 1000)

`Citizens`: germans (in 1000)

`Density`: population density 2004-12-31 (in square km)

`Pop.le.15`: population younger than (or equal to) 15 years 2002-12-31 (in percent)

`Pop.15.18`: population between 15 and 18 years old 2002-12-31 (in percent)

`Pop.18.25`: population between 18 and 25 years old 2002-12-31 (in percent)

`Pop.25.35`: population between 25 and 35 years old 2002-12-31 (in percent)

`Pop.35.60`: population between 35 and 60 years old 2002-12-31 (in percent)

`Pop.g.60`: population older than 60 years 2002-12-31 (in percent)

`Births`: live births (per 1000)

`Deaths`: deaths (per 1000)

`Move.in`: moving there in 2003 (per 1000)

`Move.out`: moving away in 2003 (per 1000)

`Increase`: increase in population (per 1000)

`Farms`: number of farms in 2001 (per 1000)

`Agriculture`: agriculturally used land (in ha)

`Mining`: mining companies and processing trade 2002-09-30 (per 1000)

`Mining.employees`: employees in mining and processing trade 2002-09-30 (per 1000)

`Apt.new`: new apartments 2002 (per 1000)

`Apt`: apartments 2002-12-31 (per 1000)

`Motorized`: motor vehicles 2003-01-31 (per 1000)

`School.finishers`: school finishers 2002 (per 1000)

`School.wo.2nd`: without secondary school (ohne Hauptschule) 2002 (in percent)

`School.2nd`: with secondary school (Hauptschule) 2002 (in percent)

`School.Real`: with graduation from Realschule 2002 (in percent)

`School.UED`: with university-entrance diploma (Gymnasium) 2002 (in percent)

`Unemployment.03`: unemployment 2003-12-31 (in percent)

`Unemployment.04`: unemployment 2004-12-31 (in percent)

`Employed`: employed subject to social insurance contribution (per 1000)

`FFF`: farmers, foresters, fishermen (in percent)

`Industry`: industry employees subject to social insurance contribution (in percent)

`CTT`: commerce, transportation and telecommunication employees subject to social insurance contribution (in percent)

`OS`: other services (in percent)

`Voters.05`: eligible voters 2005

`Voters.02`: eligible voters 2002

`Votes.05`: number of votes 2005

`Votes.02`: number of votes 2002

`Invalid.05`: invalid votes 2005

`Invalid.02`: invalid votes 2002

`Valid.05`: valid votes 2005

`Valid.02`: valid votes 2002

`Votes.SPD.05`: votes for SPD 2005

`Votes.SPD.02`: votes for SPD 2002

`Votes.CDU.CSU.05`: votes for CDU/CSU 2005

`Votes.CDU.CSU.02`: votes for CDU/CSU 2002

`Votes.Gruene.05`: votes for Gruene 2005

`Votes.Gruene.02`: votes for Gruene 2002

`Votes.FDP.05`: votes for FDP 2005

`Votes.FDP.02`: votes for FDP 2002

`Votes.Linke.05`: votes for Linke 2005

`Votes.Linke.02`: votes for Linke 2002

`SPD.05`: SPD 2005 (as a fraction in [0,1])

`CDU.CSU.05`: CDU/CSU 2005 (as a fraction in [0,1])

`Gruene.05`: Gruene 2005 (as a fraction in [0,1])

`FDP.05`: FDP 2005 (as a fraction in [0,1])

`Linke.05`: Linke 2005 (as a fraction in [0,1])

`Others.05`: Other parties 2005 (as a fraction in [0,1])

`SPD.02`: SPD 2002 (as a fraction in [0,1])

`CDU.CSU.02`: CDU/CSU 2002 (as a fraction in [0,1])

`Gruene.02`: Gruene 2002 (as a fraction in [0,1])

`FDP.02`: FDP 2002 (as a fraction in [0,1])

`Linke.02`: Linke 2002 (as a fraction in [0,1])

`Others.02`: other parties 2002 (as a fraction in [0,1])

Source

The data was obtained from http://www.bundeswahlleiter.de but is not available under this link anymore. Furthermore, the first column of the original data set is ommitted as it only contained the row numbers.

Examples

```
data("de_elect")
```

---

extract *Extracting Information from Zen Arguments*

Description

Auxiliary functions to extract information from zargs for 1d and 2d (default) plots.

Usage

```
extract_1d(zargs)
extract_2d(zargs)
```

Arguments

`zargs`	argument list as passed from `zenplot()`. This must at least contain `x`, `orientations`, `vars`, `num`, `lim` and `labs` (for `extract_1d()`) and `x`, `vars`, `num`, `lim` and `labs` (for `extract_2d()`); see `zenplot()` for an explanation of these variables.

Details

This is an auxiliary function used by the provided 1d and 2d plots. For performance reasons, no checking of the input object is done.

Value

for `extract_1d()`: `list` with the data to be plotted in the 1d plot (`x`), a list with all columns of `x` (`xcols`), the group numbers for each column of `x` (`groups`), the variable numbers for each column of `x` (`vars`), the group labels for each column of `x` (`glabs`), the variable labels for each column of `x` (`vlabs`), a `logical` indicating whether the plot is horizontal or vertical (`horizontal`) and the axis limits (`xlim`).

for `extract_2d()`: `list` with the data to be plotted in the 2d plot (`x` and `y`), a list with all columns of `x` (`xcols`), the group numbers for each column of `x` (`groups`), the variable numbers for each column of `x` (`vars`), the group labels for each column of `x` (`glabs`), the variable labels for each column of `x` (`vlabs`), the x-axis and y-axis limits (`xlim` and `ylim`) and a `logical` indicating whether the x and y variables belong to the same group (`same.group`).

Author(s)

Marius Hofert

See Also

`plots_graphics`, `plots_grid`, `plots_loon`

Examples

```
## Dummy example (mimicking how zargs are built internally)
set.seed(271)
n <- 100
x <- list(matrix(rnorm(n*2), ncol = 2), matrix(rnorm(n*3), ncol = 3))
n2dplots <- 5 - 1
pathLayout <- unfold(n2dplots)
path <- pathLayout$path
Layout <- pathLayout$layout
zargs2d <- list(x = x,
                turns = path$turns,
                orientations = Layout$orientations,
                vars = Layout$vars,
                lim = "individual",
                labs = list(group = "G", var = "V", sep = " "),
                width1d = 1,
                width2d = 10,
                num = 2,
                ispace = 0)

## Calling extract_2d()
str(extract_2d(zargs2d))
```

happiness *World Happiness Data Set*

Description

Data set consisting of 498 rows and 12 columns containing data from the World Happiness Report over three years.

Usage

```
data("happiness")
```

Format

`data.frame()` with 12 columns:

`Time`**:** year of the World Happiness Report.

`Region`**:** region of the world.

`Country`**:** country.

`Happiness`**:** happiness score measured in the respective year (see `Time`) by asking “How would you rate your happiness on a scale of 0 to 10 where 10 is happiest?”.

`Rank`**:** rank of the country based on `Happiness`.

`GDP`**:** extent to which the gross domestic product per capita contributed to the calculation of `Happiness`.

`Family`**:** extent to which family contributed to the calculation of `Happiness`.

Health: extent to which life expectancy contributed to the calculation of Happiness.

Freedom: extent to which freedom contributed to the calculation of Happiness.

Corruption: extent to which the perception of corruption contributed to the calculation of Happiness.

Generosity: extent to which generosity contributed to the calculation of Happiness.

Dystopia: extent to which the dystopia residual contributed to the calculation of Happiness. Dystopia is an imaginary country with the world's least-happy people (which can act as a benchmark against which all countries can be favorably compared).

Details

GDP, Family, Health, Freedom, Corruption and Generosity describe the extent to which these factors contribute in evaluating the happiness in each country. If added together with Dystopia, one receives the happiness score.

Source

The data set was obtained from https://www.kaggle.com/unsdsn/world-happiness on 2018-04-20 in three different .csv files (one for each year). Joint columns (variables) where then built, the rows expanded (to be the same for each year) and sorted acorrding to Region and Country. Finally, Time was added to obtain a single data set.

References

https://www.kaggle.com/unsdsn/world-happiness

Examples

```
data("happiness")
stopifnot(all.equal(rowSums(happiness[,c("GDP", "Family", "Health", "Freedom",
                                          "Corruption", "Generosity",
                                          "Dystopia")]),
                    happiness[, "Happiness"], tol = 5e-5))
```

---

occupancy *Tools for the Occupancy Matrix*

Description

Convert an occupancy matrix to matrix with different symbols.

Usage

```
convert_occupancy(x, to = c("", "<", ">", "v", "^"))
```

Arguments

`x` an occupancy `matrix` consisting of the `character` `""` (unoccupied), `"l"` (left), `"r"` (right), `"d"` (down) or `"u"` (up) as returned by `zenplot()`.

`to` a `vector` of symbols to which `""`, `"l"`, `"r"`, `"d"` and `"u"` should be mapped.

Value

`matrix` as the occupancy matrix but with entries replaced by those in `to`.

Author(s)

Marius Hofert

Examples

```
## Generate some data
n <- 1000 # sample size
d <- 20 # dimension
set.seed(271) # set seed (for reproducibility)
x <- matrix(rnorm(n * d), ncol = d) # i.i.d. N(0,1) data

## Extract the occupancy matrix from a zenplot
res <- zenplot(x)
(occ <- res[["path"]][["occupancy"]])

## Convert the occupancy matrix
convert_occupancy(occ)
```

`olive` *Olive Oil Data Set*

Description

Data set consisting of 572 rows and 10 columns containing data about olive oil.

Usage

```
data("olive")
```

Format

A `data.frame()` with 10 columns:

`area`: (larger) area.

`region`: (local) region.

`palmitic`, `palmitoleic`, `stearic`, `oleic`, `linoleic`, `linolenic`, `arachidic`, `eicosenoic`: the fatty acids measured.

Source

The data set was obtained from the package **pdfCluster** (for convenience). It contains 572 rows of observations. The first and the second column correspond to the area (Centre-North, South, Sardinia) and the geographical region of origin of the olive oils (northern Apulia, southern Apulia, Calabria, Sicily, inland Sardinia and coast Sardinia, eastern and western Liguria, Umbria), respectively. The remaining columns represent the chemical measurements (on the acid components for the oil specimens) palmitic, palmitoleic, stearic, oleic, linoleic, linolenic, arachidic, eicosenoic.

Examples

```
data("olive")
```

---

plots_graphics *Graphics-Based Plotting Functions*

Description

The 1d and 2d plotting functions based on the R package **graphics**.

Usage

```
rug_1d_graphics(zargs,
                loc = 0.5, length = 0.5, width = 1, col = par("fg"),
                add = FALSE, plot... = NULL, ...)
points_1d_graphics(zargs,
                   loc = 0.5, cex = 0.4,
                   add = FALSE, plot... = NULL, ...)
jitter_1d_graphics(zargs,
                   loc = 0.5, offset = 0.25, cex = 0.4,
                   add = FALSE, plot... = NULL, ...)
hist_1d_graphics(zargs,
                 breaks = NULL, length.out = 21, col = NULL,
                 plot... = NULL, ...)
density_1d_graphics(zargs,
                    density... = NULL, offset = 0.08,
                    add = FALSE, plot... = NULL, ...)
boxplot_1d_graphics(zargs,
                    cex = 0.4, range = NULL, axes = FALSE,
                    add = FALSE, ...)
arrow_1d_graphics(zargs,
                  loc = c(0.5, 0.5), angle = 60, length = 0.6,
                  add = FALSE, plot... = NULL, ...)
rect_1d_graphics(zargs,
                 loc = c(0.5, 0.5), width = 1, height = 1,
                 add = FALSE, plot... = NULL, ...)
lines_1d_graphics(zargs,
                  loc = c(0.5, 0.5), length = 1,
                  add = FALSE, plot... = NULL, ...)
label_1d_graphics(zargs,
                  loc = c(0.5, 0.5), label = NULL, box = FALSE,
                  add = FALSE, plot... = NULL, ...)
layout_1d_graphics(zargs, ...)


group_2d_graphics(zargs,
                  glabs = NULL, sep = "\n", loc = c(0.5, 0.5),
                  add = FALSE, plot... = NULL, ...)
points_2d_graphics(zargs,
                   cex = 0.4, box = FALSE,
                   add = FALSE, group... = NULL, plot... = NULL, ...)
qq_2d_graphics(zargs,
               do.line = TRUE, lines... = NULL, cex = 0.4, box = FALSE,
               add = FALSE, group... = NULL, plot... = NULL, ...)
density_2d_graphics(zargs,
                    ngrids = 25, drawlabels = FALSE,
                    axes = FALSE, box = FALSE,
                    add = FALSE, group... = NULL, ...)
axes_2d_graphics(zargs,
                 length = 0.1, eps = 0.04, code = 2, xpd = NA,
                 add = FALSE, group... = NULL, plot... = NULL, ...)
arrow_2d_graphics(zargs,
                  loc = c(0.5, 0.5), angle = 60, length = 0.2,
                  add = FALSE, group... = NULL, plot... = NULL, ...)
rect_2d_graphics(zargs,
                 loc = c(0.5, 0.5), width = 1, height = 1,
                 add = FALSE, group... = NULL, plot... = NULL, ...)
label_2d_graphics(zargs,
                  loc = c(0.98, 0.05), label = NULL, adj = 1:0, box = FALSE,
                  add = FALSE, group... = NULL, plot... = NULL, ...)
layout_2d_graphics(zargs, ...)
```

Arguments

`zargs` argument list as passed from `zenplot()`.

`width` width of the rugs/rectangle.

`height` height of the rugs/rectangle.

`col` color (of the rugs) or vector of colors (for the bars and bar components; see `barplot()`).

`add` `logical` indicating whether the current plot should be added to (or on top of) the previous one.

`axes` A `logical` indicating whether axes should be drawn.

`cex` character expansion factor.

`offset` number in $[0, 0.5]$ determining how far away the plot stays from the plot margins (for creating space between the two).

range argument range of the underlying boxplot() (determines how far the plot whiskers extend out of the box). If range = NULL, this will be automatically determined depending on the sample size.

breaks break points for the histogram as passed to the underlying hist(). If NULL, the default is to use 20 equi-width bins covering the range of the data.

length.out number of break points if is.null(breaks).

loc x-location or (x,y)-location (for 1d plots when viewed in the direction of the path; for 2d plots when viewed in normal viewing direction) of the center of the respective geometric shape or plot.

angle angle between the two edges of the arrow head.

length length of the arrow in $[0, 1]$ from tip to base.

label label to be used (with default being the column names of the data if NULL).

box logical indicating whether a box is drawn around the plot region.

glabs group labels being indexed by the plot variables (thus need to be of length as the number of plot variables); if NULL, they are determined with extract_2d() and the underlying burst().

sep a separator character for the group labels.

do.line logical indicating whether a Q-Q line is plotted.

lines... list of additional arguments passed to the underlying lines() function for drawing the Q-Q line through the empirical 0.25- and 0.75-quantiles.

ngrids number of grid points in each dimension (a scalar or an integer vector of length two).

drawlabels logical indicating whether the contours should be labeled.

eps distance by which the axes are moved away from the plot region.

code integer code determining the kind of arrows to be drawn; see arrows.

xpd logical or NA, determining the region with respect to which clipping takes place; see par().

adj x (and optionally y) adjustment of the label.

density... list of additional arguments passed to the underlying density().

group... list() of additional arguments passed to group_2d_graphics().

plot... list of additional arguments passed to the underlying plot().

... additional arguments passed to the underlying **graphics** functions.

Details

These functions based on the R package **graphics** are provided as useful defaults for the arguments plot1d and plot2d of zenplot(), respectively. See zenplot() for how to use them, their source code for how to adjust them or how to write your own plot1d or plot2d. The main idea is that zenplot() passes on the zargs arguments to the plot1d or plot2d functions and the ellipsis argument is used to pass down all other (mostly graphical) parameters (to both plot1d or plot2d).

Overlaying of different **graphics** functions might not always turn out nicely (e.g. arrows over a boxplot; the latter creates problems concerning the spacing). For such tasks, it is recommended to work with **grid** via pkg = "grid" in zenplot().

Value

(Mostly) `invisible()`.

Author(s)

Marius Hofert and Wayne Oldford

See Also

`zenplot()` for how to use these functions.

Examples

```
## Implementation of 1d functions (for plot1d of zenplot())
rug_1d_graphics
points_1d_graphics
jitter_1d_graphics
density_1d_graphics
boxplot_1d_graphics
hist_1d_graphics
arrow_1d_graphics
rect_1d_graphics
lines_1d_graphics
label_1d_graphics
layout_1d_graphics

## Implementation of 2d functions (for plot2d of zenplot())
group_2d_graphics
points_2d_graphics
qq_2d_graphics
density_2d_graphics
axes_2d_graphics
arrow_2d_graphics
rect_2d_graphics
label_2d_graphics
layout_2d_graphics
```

`plots_grid` *Grid-Based Plotting Functions*

Description

The 1d and 2d plotting functions based on the R package **grid**.

Usage

```
rug_1d_grid(zargs,
            loc = 0.5, length = 0.5, width = 1e-3, col = par("fg"),
            draw = FALSE, ...)
points_1d_grid(zargs,
               loc = 0.5, pch = 21, size = 0.02,
               draw = FALSE, ...)
jitter_1d_grid(zargs,
               loc = 0.5, offset = 0.25, pch = 21, size = 0.02,
               draw = FALSE, ...)
hist_1d_grid(zargs,
             breaks = NULL, length.out = 21, col = NULL, fill = NULL,
             draw = FALSE, ...)
density_1d_grid(zargs,
                density... = NULL, offset = 0.08,
                draw = FALSE, ...)
boxplot_1d_grid(zargs,
                pch = 21, size = 0.02,
                col = NULL, lwd = 2, bpwidth = 0.5, range = NULL,
                draw = FALSE, ...)
arrow_1d_grid(zargs,
              loc = c(0.5, 0.5), angle = 60, length = 0.6,
              draw = FALSE, ...)
rect_1d_grid(zargs,
             loc = c(0.5, 0.5), width = 1, height = 1,
             draw = FALSE, ...)
lines_1d_grid(zargs,
              loc = c(0.5, 0.5), length = 1, arrow = NULL,
              draw = FALSE, ...)
label_1d_grid(zargs,
              loc = c(0.5, 0.5), label = NULL, cex = 0.66,
              box = FALSE, box.width = 1, box.height = 1,
              draw = FALSE, ...)
layout_1d_grid(zargs, ...)


group_2d_grid(zargs,
              glabs = NULL, sep = "\n", loc = c(0.5, 0.5),
              draw = FALSE, ...)
points_2d_grid(zargs,
               type = c("p", "l", "o"), pch = NULL, size = 0.02,
               box = FALSE, box.width = 1, box.height = 1,
               group... = list(cex = 0.66), draw = FALSE, ...)
qq_2d_grid(zargs,
           do.line = TRUE, lines... = NULL, pch = NULL, size = 0.02,
           box = FALSE, box.width = 1, box.height = 1,
           group... = list(cex = 0.66), draw = FALSE, ...)
density_2d_grid(zargs,
                ngrids = 25, ccol = NULL, clwd = 1, clty = 1,
                box = FALSE, box.width = 1, box.height = 1,
                group... = list(cex = 0.66), draw = FALSE, ...)
axes_2d_grid(zargs,
             angle = 30, length = unit(0.05, "npc"), type = "open", eps = 0.02,
             group... = list(cex = 0.66), draw = FALSE, ...)
arrow_2d_grid(zargs,
              loc = c(0.5, 0.5), angle = 60, length = 0.2,
              group... = list(cex = 0.66), draw = FALSE, ...)
rect_2d_grid(zargs,
             loc = c(0.5, 0.5), width = 1, height = 1,
             group... = list(cex = 0.66), draw = FALSE, ...)
label_2d_grid(zargs,
              loc = c(0.98, 0.05), label = NULL, cex = 0.66,
              just = c("right", "bottom"), rot = 0,
              box = FALSE, box.width = 1, box.height = 1,
              group... = list(cex = cex), draw = FALSE, ...)
layout_2d_grid(zargs, ...)
```

Arguments

`zargs` argument list as passed from `zenplot()`.

`width` width (passed on to the underlying **grid** functions).

`height` height (passed on to the underlying **grid** functions).

`just` justification (see `rectGrob()` and `textGrob()`).

`col` for

`rug_1d_grid`**:** color and fill color of the rectangels forming the rugs.

`boxplot_1d_grid`**:** color of the box, whiskers and points.

`hist_1d_grid`**:** color of the bins.

`draw` `logical` indicating whether graphics output is produced.

`pch` plot symbol.

`size` plot symbol size as passed to `pointsGrob()`.

`offset` number in $[0, 0.5]$ determining how far away the plot stays from the plot margins (for creating space between the two).

`lwd` line width.

`bpwidth` width of the boxplot (in `default.units`).

`range` determines how far the plot whiskers extend out of the box. If `range = NULL`, this will be automatically determined depending on the sample size.

`breaks` break points for the histogram as passed to the underlying `hist()`. If `NULL`, the default is to use 20 equi-width bins covering the range of the data.

`length.out` number of break points if `is.null(breaks)`.

`fill` fill color of the bins.

`loc` (x,y)-location of the center of the arrow.

arrow see linesGrob().

label label to be used (with default being the column names of the data if NULL).

rot rotation of the label in degrees.

box logical indicating whether a box is drawn around the plot region.

box.width width of the box (if drawn).

box.height height of the box (if drawn).

cex character expansion (aims for a useful default for grid but might not always be suitable – for that one would need to know both the number of rows and columns in the plot layout and yet this would still be affected by the size of the plot window).

glabs group labels being indexed by the plot variables (thus need to be of length as the number of plot variables); if NULL, they are determined with extract_2d() and the underlying burst().

sep a separator character for the group labels.

group... list of arguments passed to group_2d_grid() (or NULL).

do.line logical indicating whether a Q-Q line is plotted.

lines... list of additional arguments passed to the underlying lines() function for drawing the Q-Q line through the empirical 0.25- and 0.75-quantiles.

ngrids number of grid points in each dimension (a scalar or an integer vector of length two).

ccol, clwd, clty

colors (col), line widths (lwd) andline types (lty) of the contour lines. These can be single values or vectors (which are then recycled).

angle angle between the two edges of the arrow head.

length length of the arrow in [0,1] from tip to base.

type axis type.

eps distance by which the axes are moved away from the plot region.

density... list() of arguments for the underlying density().

... additional (graphical) parameters passed to gpar().

Details

These functions based on the R package **grid** are provided as useful defaults for the arguments plot1d and plot2d of zenplot(), respectively. See zenplot() for how to use them, their source code for how to adjust them or how to write your own plot1d or plot2d. The main idea is that zenplot() passes on the zargs arguments to the plot1d or plot2d functions and the ellipsis argument is used to pass down all other (mostly graphical) parameters (to both plot1d or plot2d; via gpar()).

Value

(Mostly) the underlying grob via invisible().

Author(s)

Marius Hofert and Wayne Oldford

See Also

zenplot() for how to use these functions.

Examples

```
## Implementation of 1d functions (for plot1d of zenplot())
rug_1d_grid
points_1d_grid
jitter_1d_grid
density_1d_grid
boxplot_1d_grid
hist_1d_grid
arrow_1d_grid
rect_1d_grid
lines_1d_grid
label_1d_grid
layout_1d_grid

## Implementation of 2d functions (for plot2d of zenplot())
group_2d_grid
points_2d_grid
qq_2d_grid
density_2d_grid
axes_2d_grid
arrow_2d_grid
rect_2d_grid
label_2d_grid
layout_2d_grid
```

plots_loon *Loon-Based Plotting Functions*

Description

The 1d and 2d plotting functions based on the R package loon.

Usage

```
rug_1d_loon(zargs, ...)
points_1d_loon(zargs,
                linkingGroup = NULL, linkingKey = NULL,
                showLabels = FALSE, showScales = FALSE,
                showGuides = FALSE, glyph = "ocircle",
                itemLabel = NULL, showItemLabels = TRUE,
                parent = NULL, ...)
```

```
jitter_1d_loon(zargs,
               linkingGroup = NULL, showLabels = FALSE,
               showScales = FALSE, showGuides = FALSE,
               glyph = "ocircle", itemLabel = NULL,
               showItemLabels = TRUE, parent = NULL, ...)
hist_1d_loon(zargs,
             breaks = NULL, color = NULL, fill = NULL,
             showStackedColors = TRUE,
             showBinHandle = FALSE, showLabels = FALSE,
             linkingGroup = NULL, showScales = FALSE,
             showGuides = FALSE, parent = NULL, ...)
density_1d_loon(zargs,
                density.args = list(), method = c("single", "double"),
                lwd = NULL, linewidth = NULL, color = NULL,
                fill = NULL, linecolor = NULL, linkingGroup = NULL,
                showLabels = FALSE, showScales = FALSE,
                showGuides = FALSE, baseplot = NULL, parent = NULL, ...)
boxplot_1d_loon(zargs,
                color = NULL, linecolor = NULL, lwd = 2,
                range = NULL, showLabels = FALSE, showScales = FALSE,
                showGuides = FALSE, linkingGroup = NULL,
                baseplot = NULL, parent, ...)
arrow_1d_loon(zargs,
              loc = c(0.5, 0.5), length = 0.6, angle = NULL,
              linkingGroup = NULL, showLabels = FALSE,
              showScales = FALSE, showGuides = FALSE,
              baseplot = NULL, parent = NULL, ...)
rect_1d_loon(zargs,
             loc.x = NULL, loc.y = NULL, color = NULL,
             fill = NULL, lwd = 1,
             linkingGroup = NULL, showLabels = FALSE,
             showScales = FALSE, showGuides = FALSE,
             baseplot = NULL, parent = NULL, ...)
lines_1d_loon(zargs,
              loc.x = NULL, loc.y = NULL,
              color = NULL, lwd = 1,
              linkingGroup = NULL,
              showLabels = FALSE, showScales = FALSE,
              showGuides = FALSE, baseplot = NULL,
              parent = NULL, ...)
label_1d_loon(zargs,
              loc.x = NULL, loc.y = NULL, label = NULL,
              rot = NULL, size = 8, box = FALSE, color = NULL,
              linkingGroup = NULL, showLabels = FALSE,
              showScales = FALSE, showGuides = FALSE,
              baseplot = NULL, parent = NULL, ...)
layout_1d_loon(zargs, ...)
```

```
group_2d_loon(zargs,
              glabs = NULL, sep = "\n", size = 8, rot = 0,
              baseplot = NULL, parent = NULL, ...)
points_2d_loon(zargs,
               showLabels = FALSE, showScales = FALSE,
               showGuides = FALSE, linkingGroup = NULL,
               linkingKey = NULL, glyph = "ocircle",
               itemLabel = NULL, showItemLabels = TRUE,
               parent = NULL, group... = NULL, ...)
density_2d_loon(zargs, ngrids = 25,
                ccol = NULL, color = NULL, clwd = NULL, lwd = NULL,
                linewidth = 1, showLabels = FALSE,
                showScales = FALSE, showGuides = FALSE,
                linkingGroup = NULL,
                baseplot = NULL, parent = NULL, group... = NULL, ...)
axes_2d_loon(zargs,
             angle = 30, length = 0.05, eps = 0.02,
             linkingGroup = NULL, color = NULL, showLabels = FALSE,
             showScales = FALSE, showGuides = FALSE,
             baseplot = NULL, parent = NULL,
             group... = NULL, ...)
arrow_2d_loon(zargs,
              loc = rep(0.5, 2), length = 0.2, angle = 30,
              linkingGroup = NULL, color = NULL,
              showLabels = FALSE, showScales = FALSE,
              showGuides = FALSE, baseplot = NULL, parent = NULL,
              group... = NULL, ...)
rect_2d_loon(zargs, loc.x = NULL, loc.y = NULL, color = NULL,
             fill = NULL, lwd = 1, linkingGroup = NULL,
             showLabels = FALSE, showScales = FALSE,
             showGuides = FALSE, baseplot = NULL,
             parent = NULL, group... = NULL, ...)
label_2d_loon(zargs,
              loc = NULL, label = NULL, rot = 0, size = 8,
              box = FALSE, color = NULL,
              linkingGroup = NULL, showLabels = FALSE,
              showScales = FALSE, showGuides = FALSE,
              baseplot = NULL, parent = NULL,
              group... = NULL, ...)
layout_2d_loon(zargs, ...)
```

Arguments

zargs	argument list as passed from zenplot().
linkingGroup	string specifying the initial group of plots to be linked to this plot.
linkingKey	list of IDs to link on.
showLabels	logical determining whether axis labels are displayed.

showScales logical determining whether scales are displayed.

showGuides logical determining whether the background guidelines are displayed.

glyph string determining the glyph type to be displayed for points, default is an open circle: "ocircle".

itemLabel vector of strings to serve as the item label.

showItemLabels logical determing whether item labels display on mouse hover.

parent tk parent for this loon plot widget.

density.args list() of arguments for density().

method type of density plot used (single or double; the latter being reflected).

lwd line width.

linewidth for

- **density_1d_loon:** line width of outline for density polygons (highest priority).
- **density_2d_loon:** line width used when both clwd and lwd are NULL, value of 1 used otherwise.

linecolor for

- **density_1d_loon:** colour used for the outline of the density.
- **boxplot_1d_loon:** colour used for the lines to draw the boxplot.

color color.

fill fill color.

baseplot if non-null the base plot on which the plot should be layered.

range determines how far the plot whiskers extend out of the box. If range = NULL, this will be automatically determined depending on the sample size.

breaks break points for the histogram as passed to the underlying hist(). If NULL, the default is to use 20 equi-width bins covering the range of the data.

showStackedColors
logical determining whether to show the individual point colours stacked in the histogram.

showBinHandle logical to show a handle to adjust bins.

loc location of the center of the arrow.

length length of the arrow head.

angle angle between the shaft and one of the edges of the arrow head.

loc.x, loc.y x- and y-location.

label label to be used (with a useful default if NULL).

rot rotation of the label in degrees.

size plot size.

box logical indicating whether a box is drawn around the plot.

glabs group labels being indexed by the plot variables (thus need to be of length as the number of plot variables); if NULL, they are determined with extract_2d() and the underlying burst().

`sep` a separator `character` for the group labels.

`group...` list of arguments passed to `group_2d_loon()` (or `NULL`).

`ngrids` number of grid points in each dimension (a scalar or an integer vector of length two).

`ccol, clwd` colors (`ccol`) and line widths (`clwd`) of the contour lines. These can be single values or vectors (which are then recycled).

`eps` distance by which the axes are moved away from the plot region.

`...` additional arguments passed to the underlying loon functions.

Details

These functions based on the R package loon are provided as useful choices for the arguments `plot1d` and `plot2d` of `zenplot()`. See `zenplot()` how to use them, their source code for how to adjust them, their source code for how to adjust them or how towrite your own `plot1d` or `plot2d`. The main idea is that `zenplot()` passes on the `zargs` arguments to the `plot1d` or `plot2d` functions and the ellipsis argument is used to pass down all other (mostly graphical) parameters (to both `plot1d` or `plot2d`).

Value

The path, layout and a loon object (returned invisibly).

Author(s)

Marius Hofert and Wayne Oldford

See Also

`zenplot()` for how to use these functions.

Examples

```
## Implementation of 1d functions (for plot1d of zenplot())
rug_1d_loon
points_1d_loon
jitter_1d_loon
density_1d_loon
boxplot_1d_loon
hist_1d_loon
arrow_1d_loon
rect_1d_loon
lines_1d_loon
label_1d_loon
layout_1d_loon

## Implementation of 2d functions (for plot2d of zenplot())
group_2d_loon
points_2d_loon
density_2d_loon
axes_2d_loon
```

```
arrow_2d_loon
rect_2d_loon
label_2d_loon
layout_2d_loon
```

plot_indices *Plot Indices of the Current Plot*

Description

Determining the indices of the x and y variables of the current plot.

Usage

```
plot_indices(zargs)
```

Arguments

zargs	argument list as passed from zenplot(). This must at least contain vars and num; see zenplot() for an explanation of these variables.

Details

This is an auxiliary function useful, for example, when writing user-provided 1d and 2d plot functions.

Value

A numeric(2) containing the indices of the x and y variables to be plotted in the current plot (the plot with number num). If the current plot is a 2d plot, the same variable is used twice.

Author(s)

Marius Hofert

Examples

```
plot_indices # its definition
```

`plot_region` *Setting up Plot Region for Graphics Functions*

Description

Auxiliary function for setting up the plot region of 1d and 2d graphics plots.

Usage

```
plot_region(xlim, ylim, plot... = NULL)
```

Arguments

`xlim`	x-axis limits.
`ylim`	y-axis limits.
`plot...`	arguments passed to `plot()`.

Details

This is an auxiliary function used by the provided **graphics**-related 1d and 2d plots.

Value

`invisible()`.

Author(s)

Marius Hofert

See Also

plots_graphics

Examples

```
plot_region
```

`vport` *Viewport Constructing Function for Grid Functions*

Description

Auxiliary function for constructing viewports for 1d and 2d (default) plots.

Usage

```
vport(ispace, xlim = NULL, ylim = NULL, x = NULL, y = NULL, ...)
```

Arguments

`ispace`	inner space (in $[0, 1]$).
`xlim`	x-axis limits; if `NULL`, the data limits are used.
`ylim`	y-axis limits; if `NULL`, the data limits are used.
`x`	x data (only used if `is.null(xlim)`); if `NULL`, `0:1` is used.
`y`	y data (only used if `is.null(ylim)`); if `NULL`, `0:1` is used.
`...`	additional arguments passed to the underlying `viewport()`.

Details

This is an auxiliary function used by the provided **grid**-related 1d and 2d plots.

Value

A `viewport`.

Author(s)

Marius Hofert

See Also

plots_grid

Examples

```
vport
```

wine *Wine Data Set*

Description

Data set consisting of 178 rows and 27 columns containing data about wine from the Piedmont region of Italy.

Usage

```
data("wine")
```

Format

`data.frame()` with 27 columns:

`wine`: wine name (categorical variable with levels `Barbera`, `Barolo`, `Grignolino`).
`alcohol`: alcohol percentage (numeric).
`sugar`: sugar-free extract (numeric).
`acidity`: fixed acidity (numeric).
`tartaric`: tartaric acid (numeric).
`malic`: malic acid (numeric).
`uronic`: uronic acids (numeric).
`pH`: pH (numeric).
`ash`: ash (numeric).
`alcal_ash`: alcalinity of ash (numeric).
`potassium`: potassium (numeric).
`calcium`: calcium (numeric).
`magnesium`: magnesium (numeric).
`phosphate`: phosphate (numeric).
`cloride`: chloride (numeric).
`phenols`: total phenols (numeric).
`flavanoids`: flavanoids (numeric).
`nonflavanoids`: nonflavanoid phenols (numeric).
`proanthocyanins`: proanthocyanins (numeric).
`colour`: colour intensity (numeric).
`hue`: hue (numeric).
`OD_dw`: OD_{280}/OD_{315} of diluted wines (numeric).
`OD_fl`: OD_{280}/OD_{315} of flavanoids (numeric).
`glycerol`: glycerol (numeric).
`butanediol`: 2,3-butanediol (numeric).
`nitrogen`: total nitrogen (numeric).
`proline`: proline (numeric).
`methanol`: methanol (numeric).

Source

The data set was obtained from the R\ package **sn** (for convenience). It represent chemical measurements on each of 178 wine specimens belonging to three types of wine produced in the Piedmont region of Italy. The data set includes all variables listed by Forina *et al.* (1986) with the exception of 'Sulphate'. The first variable is categorial, all others are numeric.

Forina, M., Lanteri, S. Armanino, C., Casolino, C., Casale, M. and Oliveri, P. V-PARVUS 2008: an extendible package of programs for esplorative data analysis, classification and regression analysis. Dip. Chimica e Tecnologie Farmaceutiche ed Alimentari, Università di Genova, Italia. Web-site (not accessible as of 2014): 'http://www.parvus.unige.it'

References

Forina M., Armanino C., Castino M. and Ubigli M. (1986). Multivariate data analysis as a discriminating method of the origin of wines. *Vitis* **25**, 189–201.

Examples

```
data("wine")
```

---

zenpath *Constructing Zenpaths and Related Tools*

Description

Constructing zenpaths and tools for extracting, connecting and displaying pairs, as well as, grouping and indexing data structures.

Usage

```
zenpath(x, pairs = NULL,
        method = c("front.loaded", "back.loaded", "balanced",
                   "eulerian.cross", "greedy.weighted", "strictly.weighted"),
        decreasing = TRUE)
extract_pairs(x, n)
connect_pairs(x, duplicate.rm = FALSE)
graph_pairs(x, var.names = NULL)
groupData(x, indices, byrow = FALSE)
indexData(x, indices)
```

Arguments

`x` for

`zenpath()`: for method

- `"front.loaded"`: single `integer`.
- `"back.loaded"`: as for `method = "front.loaded"`.
- `"balanced"`: as for `method = "front.loaded"`.

"eulerian.cross": two integers representing the group sizes.

"greedy.weighted": numeric weight vector (or matrix or distance matrix).

"strictly.weighted": as for method = "greedy.weighted".

extract_pairs(): the path, a vector or list of indices of the variables to be plotted.

connect_pairs(): two-column matrix or a list containing vectors of length two representing the pairs to be connected.

graph_pairs(): matrix or list of pairs along a zenpath. Can also be a list containing vectors of length larger than two (then being interpreted as connected pairs).

groupData(): matrix (or an object convertible to such via as.matrix()).

indexData(): matrix or data.frame (most useful for the latter).

pairs
: two-column matrix containing (row-wise) the pairs of connected variables to be sorted according to the weights. pairs is only used for methods greedy.weighted, strictly.weighted and can be NULL in which case a default is constructed in lexicographical order.

method
: character string indicating the sorting method to be used. Available are:

"front.loaded": sort all pairs such that the first variables appear the most frequently early in the sequence.

"back.loaded": sort all pairs such that the later variables appear the most frequently later in the sequence.

"balanced": sort all pairs such that all variables appear in balanced blocks throughout the sequence (a Hamiltonian Decomposition).

"eulerian.cross": generate a sequence of pairs such that each is formed with one variable from each group.

"greedy.weighted": sort all pairs according to a greedy (heuristic) Euler path visiting each edge precisely once.

"strictly.weighted": this method strictly respects the order given by the weights, so the first, second, third, etc. adjacent pair of numbers of the output of zenpath() corresponds to the pair with largest, second-largest, third-largest, etc. weight.

decreasing
: logical indicating whether the sorting is done according to increasing or decreasing weights.

n
: vector of length two giving the number of pairs to extract from the path x (if NULL, all pairs are returned (nothing extracted); if of length one, it is replicated). The first number corresponds to the beginning of the path, the second to the end; at least one of the two numbers should be >= 1.

duplicate.rm
: logical indicating whether equal pairs (up to permutation) are omitted.

var.names
: names of the variables appearing in x.

indices
: **groupData():** list of vectors of indices according to which x is grouped.

indexData(): vector of column indices of x (typically obtained from zenpath()).

byrow
: logical indicating whether the grouping is done by row (byrow = TRUE) or by column (byrow = FALSE).

Value

zenpath() returns a sequence of variables (indices or names, possibly a list of such), which can then be used to index the data (via groupData()) for plotting via zenplot().

extract_pairs() returns an object of the same type as the input x but (possibly) shortened. It extracts the first/last so-many pairs of x.

connect_pairs() returns a list of (possibly connected) pairs, so a list of vectors of length at least 2.

groupData() returns a list of (grouped) matrices. This is then typically passed on to zenplot().

indexData() returns an object as x (typically a data.frame or matrix) containing x indexed by indices.

Author(s)

Marius Hofert and Wayne Oldford

See Also

zenplot() which provides the zenplot.

Examples

```
## A baby example to see how groupData() works
A <- matrix(1:12, ncol = 3)
lst <- list(1, list(2:3))
groupData(A, indices = lst) # split the matrix according to the grouping given by lst

## Some calls of zenpath()
zenpath(10) # integer argument
## Note that the result is of length 50 > 10 choose 2 as the underlying graph has to
## be even (and thus edges are added here)
(zp <- zenpath(c(3, 5), method = "eulerian.cross")) # integer(2) argument

## Extract the first and last three pairs of indices
extract_pairs(zp, n = 3)

## A more sophisticated example
nVars <- 5 # number of variables
set.seed(271)
x <- runif(nVars*(nVars-1)/2) # weights
## Construct the pairs
pairs <- expand.grid(1:nVars, 1:nVars)[,2:1]
pairs <- pairs[pairs[,1] < pairs[,2],]
pairs <- matrix(unlist(pairs), ncol = ncol(pairs))
stopifnot(length(x) == nrow(pairs)) # sanity check
## Manually compute the result of method = "strictly.weighted" and group the pairs
## 1) Sort pairs according to the weights x and plot the variables
w <- order(x, decreasing = TRUE)
(pairs. <- pairs[w,])
library(graph)
plot(graph_pairs(pairs.)) # depict all pairs (edge = pair)
```

```
## 2) Now go through the rows and determine the sequence of adjacent pairs
##    which can be plotted with a zenplot
res <- list(c(5,3,1),
            c(3,2,5),
            c(4,1,5),
    c(1,2),
    c(5,4,3),
    c(2,4))
## Call zenpath() and check whether we get the same
(zp  <- connect_pairs(zenpath(x, pairs = pairs, method = "strictly.weighted")))
stopifnot(identical(zp, lapply(res, as.integer)))

## Extract the first and last three pairs of indices
(ezp <- extract_pairs(zp, n = 3))

## Another example based on a matrix of (trivial) weights
## This also shows that an input matrix 'x' does not have to
## be symmetric. In that case, the lower triangular matrix is used.
d <- 10
x <- matrix(1, nrow = d, ncol = d)
k <- 1
for(j in 1:(d-1)) {
    for(i in (j+1):d) {
        x[i,j] <- k
        k <- k+1
    }
}
x

## Compute the 'strictly.weighted' zenpath (all pairs sorted in decreasing order)
k <- 10 # bottom and top number of pairs (k most extreme pairs)
zpath <- zenpath(x, method = "strictly.weighted") # compute path over all pairs (decreasing weights)
stopifnot(sapply(1:length(zpath), function(i) x[zpath[[i]][1], zpath[[i]][2]]) ==
          45:1) # check
zpath <- connect_pairs(zpath) # connect the pairs
zp <- extract_pairs(zpath, n = c(3, 0)) # grab out the top three pairs
```

zenplot *Zigzag Expanded Navigation Plots*

Description

Construct and draw a zigzag expanded navigation plot for a graphical exploratory analysis of a path of variables.

Usage

```
unfold(nfaces, turns = NULL,
       n2dcols = c("letter", "square", "A4", "golden", "legal"),
       method = c("tidy", "double.zigzag", "single.zigzag", "rectangular"),
           first1d = TRUE, last1d = TRUE, width1d = 1, width2d = 10)
zenplot(x, turns = NULL, first1d = TRUE, last1d = TRUE,
        n2dcols = c("letter", "square", "A4", "golden", "legal"),
        n2dplots = NULL,
        plot1d = c("label", "points", "jitter", "density", "boxplot", "hist",
                   "rug", "arrow", "rect", "lines", "layout"),
       plot2d = c("points", "density", "axes", "label", "arrow", "rect", "layout"),
        zargs = c(x = TRUE, turns = TRUE, orientations = TRUE,
                  vars = TRUE, num = TRUE, lim = TRUE, labs = TRUE,
                  width1d = TRUE, width2d = TRUE,
                  ispace = match.arg(pkg) != "graphics"),
        lim = c("individual", "groupwise", "global"),
        labs = list(group = "G", var = "V", sep = ", ", group2d = FALSE),
        pkg = c("graphics", "grid", "loon"),
        method = c("tidy", "double.zigzag", "single.zigzag", "rectangular"),
        width1d = if(is.null(plot1d)) 0.5 else 1, width2d = 10,
        ospace = if(pkg == "loon") 0 else 0.02,
        ispace = if(pkg == "graphics") 0 else 0.037,
        draw = TRUE, ...)
```

Arguments

nfaces number of faces of the hypercube to unfold.

x data object, typically a vector, matrix, data.frame, or a list of such. In case of a list, the components of x are interpreted as groups of data which are visually separated by a two-dimensional (group) plot.

turns character vector (of length two times the number of variables to be plotted minus 1) consisting of "d", "u", "r" or "l" indicating the turns out of the current plot position; if NULL, the turns are constructed (if x is of the form described before).

n2dcols number of columns of 2d plots (≥ 1) or one of "letter", "square", "A4", "golden" or "legal" in which case a similar layout is constructed. Note that n2dcols is ignored if !is.null(turns).

n2dplots number of 2d plots.

plot1d function returning a one-dimensional plot constructed with package pkg. Alternatively, a character string of an existing function. For the defaults provided, the corresponding functions are obtained when appending _1d_graphics, _1d_grid or _1d_loon depending on which pkg is used. Another feature is plot1d = NULL in which case no plot is constructed.

plot2d function returning a two-dimensional plot constructed with package pkg. Alternatively, a character string of an existing function. For the defaults provided, the corresponding functions are obtained when appending _2d_graphics, _2d_grid or _2d_loon depending on which pkg is used. As for plot1d, plot2d allows for plot2d = NULL.

first1d logical indicating whether the first one-dimensional plot is included.

last1d logical indicating whether the last one-dimensional plot is included.

`zargs` fully named `logical` `vector` indicating whether the respective arguments are (possibly) passed to `plot1d()` and `plot2d()` (if the latter contain the formal argument `zargs`, which they typically do/should, but see below for an example in which they do not). `zargs` can maximally contain all variables as given in the default. If one of those variables does not appear in `zargs`, it is treated as `TRUE` and the corresponding arguments are passed on to `plot1d` and `plot2d`. If one of them is set to `FALSE`, the argument is not passed on.

`lim` (x-/y-)axis limits. This can be a `character` string or a `numeric(2)`. If `lim = "groupwise"` and `x` does not contain groups, the behavior is equivalent to `lim = "global"`.

`labs` plot labels to be used; see the argument `labs` of `burst()` for the exact specification. Can, in general, be anything as long as `plot1d` and `plot2d` know how to deal with it.

`pkg` R package used for plotting (depends on how the functions `plot1d` and `plot2d` were constructed; the user is responsible for choosing the appropriate package among the supported ones).

`method` type of zigzag plot (a `character`). Available are:

- `tidy`**:** more tidied-up `double.zigzag` (slightly more compact placement of plots towards the end).
- `double.zigzag`**:** zigzag plot in the form of a flipped “S”. Along this path, the plots are placed in the form of an “S” which is rotated counterclockwise by 90 degrees.
- `single.zigzag`**:** zigzag plot in the form of a flipped “S”.
- `rectangular`**:** plots that fill the page from left to right and top to bottom. This is useful (and most compact) for plots that do not share an axis.

Note that `method` is ignored if `turns` are provided.

`width1d` graphical parameter > 0 giving the width of 1d plots.

`width2d` graphical parameter > 0 giving the width of 2d plots.

`ospace` vector being repeated to have length four giving the (bottom, left, top, right) outer space between the device region and the inner plot region in $[0, 1]$ around the zenplot.

`ispace` vector being repeated to have length four giving the (bottom, left, top, right) inner space between the figure region and the plot region in $[0, 1]$.

`draw` `logical` indicating whether a plot is created.

`...` additional arguments passed to both `plot1d` and `plot2d`. If you need to pass certain arguments only to one of them, say, `plot2d`, consider providing your own `plot2d`; see the examples below.

Value

`unfold()` returns a `list` consisting of the path (itself a `list` containing turns (a `character` vector with elements in “l”, “r”, “d”, “u”), positions (a 2-column `matrix` of (x,y)-indices in the occupancy matrix) and the occupancy matrix itself (a `matrix` with elements in 0–4 where 0 stands for “not occupied” and 1–4 encode “l”, “r”, “d”, “u”)) and details about the layout (another `list`).

`zenplot()` (besides plotting) invisibly returns a list containing the path and layout. For `pkg = "grid"`, the whole plot as a `grob` (grid object) is returned additionally. For `pkg = "loon"`, the whole plot as a loon object and the toplevel tk object is returned additionally.

Author(s)

Marius Hofert and Wayne Oldford

See Also

All provided default plot1d and plot2d functions, see plots_graphics, plots_grid, plots_loon.

extract_1d() and extract_2d() for how zargs can be split up into a list of columns and corresponding group and variable information.

burst() for how x can be split up into all sorts of information useful for plotting (see our default plot1d and plot2d).

vport() for how to construct a viewport for (our default) **grid** (plot1d and plot2d) functions.

extract_pairs(), connect_pairs(), group() and zenpath() for (zen)path-related functions.

The various vignettes for additional examples.

Examples

```
### Basics #####################################################################

## Generate some data
n <- 1000 # sample size
d <- 20 # dimension
set.seed(271) # set seed (for reproducibility)
x <- matrix(rnorm(n * d), ncol = d) # i.i.d. N(0,1) data

## A basic zenplot
res <- zenplot(x)
stopifnot(identical(res, unfold(nfaces = d - 1)))
## => The return value of zenplot() is the underlying unfold()

## Some missing data
z <- x
z[seq_len(n-10), 5] <- NA # all NA except 10 points
zenplot(z)

## Another column with fully missing data (use arrows)
## Note: This could be more 'compactified', but is technically
##       more involved
z[, 6] <- NA # all NA
zenplot(z)

## Lists of vectors, matrices and data frames as arguments (=> groups of data)
## Only two vectors
z <- list(x[,1], x[,2])
zenplot(z)

## A matrix and a vector
z <- list(x[,1:2], x[,3])
zenplot(z)

## A matrix, NA column and a vector
```

```
z <- list(x[,1:2], NA, x[,3])
zenplot(z)
z <- list(x[,1:2], cbind(NA, NA), x[,3])
zenplot(z)
z <- list(x[,1:2], 1:10, x[,3])
zenplot(z)

## Without labels or with different labels
z <- list(A = x[,1:2], B = cbind(NA, NA), C = x[,3])
zenplot(z, labs = NULL) # without any labels
zenplot(z, labs = list(group = NULL, group2d = TRUE)) # without group labels
zenplot(z, labs = list(group = NULL)) # without group labels unless groups change
zenplot(z, labs = list(var = NULL)) # without variable labels
zenplot(z, labs = list(var = "Variable ", sep = " - ")) # change default labels

## Example with a factor
zenplot(iris)
zenplot(iris, lim = "global") # global scaling of axis
zenplot(iris, lim = "groupwise") # acts as 'global' here (no groups in the data)


### More sophisticated examples ################################################

## Note: The third component (data.frame) naturally has default labels.
##       zenplot() uses these labels and prepends a default group label.
z <- list(x[,1:5], x[1:10, 6:7], NA,
          data.frame(x[seq_len(round(n/5)), 8:19]), cbind(NA, NA), x[1:10, 20])
zenplot(z, labs = list(group = "Group ")) # change the group label (var and sep are defaults)
## Alternatively, give z labels
names(z) <- paste("Group", LETTERS[seq_len(length(z))]) # give group names
zenplot(z) # uses given group names
## Now let's change the variable labels
z. <- lapply(z, function(z.) {
    if(!is.matrix(z.)) z. <- as.matrix(z.)
    colnames(z.) <- paste("Var.", seq_len(ncol(z.)))
    z.
})
zenplot(z.)


### A dynamic plot based on 'loon' (if installed and R compiled with tcl support)


if(requireNamespace("loon", quietly = TRUE))
    zenplot(x, pkg = "loon")




### Providing your own turns ###################################################

## A basic example
turns <- c("l","d","d","r","r","d","d","r","r","u","u","r","r","u","u","l","l",
           "u","u","l","l","u","u","l","l","d","d","l","l","d","d","l","l",
           "d","d","r","r","d","d")
zenplot(x, plot1d = "layout", plot2d = "layout", turns = turns) # layout of plot regions
## => The tiles stick together as ispace = 0.
zenplot(x, plot1d = "layout", plot2d = "layout", turns = turns,
        pkg = "grid") # layout of plot regions with grid
## => Here the tiles show the small (default) ispace

## Another example (with own turns and groups)
zenplot(list(x[,1:3], x[,4:7]), plot1d = "arrow", plot2d = "rect",
        turns = c("d", "r", "r", "r", "r", "d",
                  "d", "l", "l", "l", "l", "l"), last1d = FALSE)


### Providing your own plot1d() or plot2d() ####################################

## Creating a box
zenplot(x, plot1d = "label", plot2d = function(zargs)
    density_2d_graphics(zargs, box = TRUE))

## With grid

zenplot(x, plot1d = "label", plot2d = function(zargs)
        density_2d_grid(zargs, box = TRUE), pkg = "grid")


## An example with width1d = width2d and where no zargs are passed on.
## Note: This could have also been done with 'rect_2d_graphics(zargs, col = ...)'
##       as plot1d and plot2d.
myrect <- function(...) {
    plot(NA, type = "n", ann = FALSE, axes = FALSE, xlim = 0:1, ylim = 0:1)
    rect(xleft = 0, ybottom = 0, xright = 1, ytop = 1, ...)
}
zenplot(matrix(0, ncol = 15),
        n2dcol = "square", width1d = 10, width2d = 10,
        plot1d = function(...) myrect(col = "royalblue3"),
        plot2d = function(...) myrect(col = "maroon3"))

## Colorized rugs as plot1d()
basecol <- c("royalblue3", "darkorange2", "maroon3")
palette <- colorRampPalette(basecol, space = "Lab")
cols <- palette(d) # different color for each 1d plot
zenplot(x, plot1d = function(zargs)
    rug_1d_graphics(zargs, col = cols[(zargs$num+1)/2]))

## With grid
library(grid) # for gTree() and gList()

zenplot(x, pkg = "grid", # you are responsible for choosing the right pkg (cannot be tested!)
        plot1d = function(zargs)
               rug_1d_grid(zargs, col = cols[(zargs$num+1)/2]))


## Rectangles with labels as plot2d() (shows how to overlay plots)
## With graphics
## Note: myplot2d() could be written directly in a simpler way, but is
##       based on the two functions here to show how they can be combined.
zenplot(x, plot1d = "arrow", plot2d = function(zargs) {
    rect_2d_graphics(zargs)
    label_2d_graphics(zargs, add = TRUE)
})

## With grid

zenplot(x, pkg = "grid", plot1d = "arrow", plot2d = function(zargs)
    gTree(children = gList(rect_2d_grid(zargs),
                           label_2d_grid(zargs))))


## Rectangles with labels outside the 2d plotting region as plot2d()
## With graphics
zenplot(x, plot1d = "arrow", plot2d = function(zargs) {
    rect_2d_graphics(zargs)
    label_2d_graphics(zargs, add = TRUE, xpd = NA, srt = 90,
                      loc = c(1.04, 0), adj = c(0,1), cex = 0.7)
})

## With grid

zenplot(x, pkg = "grid", plot1d = "arrow", plot2d = function(zargs)
    gTree(children = gList(rect_2d_grid(zargs),
                           label_2d_grid(zargs, loc = c(1.04, 0),
                                         just = c("left", "top"),
                                         rot = 90, cex = 0.45))))


## 2d density with points, 1d arrows and labels
zenplot(x, plot1d = function(zargs) {
        rect_1d_graphics(zargs)
        arrow_1d_graphics(zargs, add = TRUE, loc = c(0.2, 0.5))
        label_1d_graphics(zargs, add = TRUE, loc = c(0.8, 0.5))
    }, plot2d = function(zargs) {
        points_2d_graphics(zargs, col = adjustcolor("black", alpha.f = 0.4))
        density_2d_graphics(zargs, add = TRUE)
})

## 2d density with labels, 1d histogram with density and label
## Note: The 1d plots are *improper* overlays here as the density
##       plot does not know the heights of the histogram. In other
##       words, both histograms and densities use the whole 1d plot
##       region but are not correct relative to each other in the
##       sense of covering the same are. For a *proper* overlay
##       see below.
zenplot(x, plot1d = function(zargs) {
        hist_1d_graphics(zargs)
        density_1d_graphics(zargs, add = TRUE, border = "royalblue3", lwd = 1.4)
        label_1d_graphics(zargs, add = TRUE, loc = c(0.2, 0.8), cex = 0.6, font = 2,
                          col = ”darkorange2”)
}, plot2d = function(zargs) {
    density_2d_graphics(zargs)
    points_2d_graphics(zargs, add = TRUE,
                       col = adjustcolor(”black”, alpha.f = 0.3))
})


### More sophisticated examples ################################################

### Example: Overlaying histgrams with densities (the *proper* way)

## Define proper 1d plot for overlaying histograms with densities
hist_with_density_1d <- function(zargs)
{
    ## Extract information and data
    num <- zargs$num # plot number (among all 1d and 2d plots)
    turn.out <- zargs$turns[num] # turn out of current position
    horizontal <- turn.out == ”d” || turn.out == ”u”
    ii <- plot_indices(zargs) # the indices of the 'x' variable to be displayed in the current plot
    label <- paste0(”V”, ii[1]) # label
    srt <- if(horizontal) 0 else if(turn.out == ”r”) -90 else 90 # label rotation
    x <- zargs$x[,ii[1]] # data
    lim <- range(x) # data limits
    ## Compute histogram information
    breaks <- seq(from = lim[1], to = lim[2], length.out = 21)
    binInfo <- hist(x, breaks = breaks, plot = FALSE)
    binBoundaries <- binInfo$breaks
    widths <- diff(binBoundaries)
    heights <- binInfo$density
    ## Compute density information
    dens <- density(x)
    xvals <- dens$x
    keepers <- (min(x) <= xvals) & (xvals <= max(x)) # keep those within the range of the data
    x. <- xvals[keepers]
    y. <- dens$y[keepers]
    ## Determine plot limits and data
    if(turn.out == ”d” || turn.out == ”l”) { # flip density/histogram
        heights <- -heights
        y. <- -y.
    }
    if(horizontal) {
        xlim <- lim
        xlim.bp <- xlim - xlim[1] # special for barplot(); need to shift the bars
        ylim <- range(0, heights, y.)
        ylim.bp <- ylim
        x <- c(xlim[1], x., xlim[2]) - xlim[1] # shift due to plot region set up by barplot()
        y <- c(0, y., 0)
    } else {
        xlim <- range(0, heights, y.)
        xlim.bp <- xlim
        ylim <- lim
        ylim.bp <- ylim - ylim[1] # special for barplot(); need to shift the bars
        x <-  c(0, y., 0)
      y <- c(xlim[1], x., xlim[2]) - ylim[1] # shift due to plot region set up by barplot()
    }
    ## Determining label position relative to the zenpath
    loc <- c(0.1, 0.6)
   if(turn.out == "d") loc <- 1-loc # when walking downwards, change both left/right and up/down
   if(turn.out == "r") { # when walking to the right, coordinates change and 2nd is flipped
        loc <- rev(loc)
        loc[2] <- 1-loc[2]
    }
   if(turn.out == "l") { # when walking to the left, coordinates change and 1st is flipped
        loc <- rev(loc)
        loc[1] <- 1-loc[1]
    }
    ## Plotting
    barplot(heights, width = widths, xlim = xlim.bp, ylim = ylim.bp,
           space = 0, horiz = !horizontal, main = "", xlab = "", axes = FALSE) # histogram
    polygon(x = x, y = y, border = "royalblue3", lwd = 1.4) # density
    opar <- par(usr = c(0, 1, 0, 1)) # switch to relative coordinates for text
    on.exit(par(opar))
    text(x = loc[1], y = loc[2], labels = label, cex = 0.7, srt = srt, font = 2,
         col = "darkorange2") # label
}

## Zenplot
zenplot(x, plot1d = "hist_with_density_1d",
        plot2d = function(zargs) {
            density_2d_graphics(zargs)
            points_2d_graphics(zargs, add = TRUE,
                               col = adjustcolor("black", alpha.f = 0.3))
})


### Example: A path through pairs of a grouped t copula sample


## 1) Build a random sample from a 17-dimensional grouped t copula
d. <- c(8, 5, 4) # sector dimensions
d <- sum(d.) # total dimension
nu <- rep(c(12, 1, 0.25), times = d.) # d.o.f. for each dimension
n <- 500 # sample size
set.seed(271)
Z <- matrix(rnorm(n * d), ncol = n) # (d,n)-matrix
P <- matrix(0.5, nrow = d, ncol = d)
diag(P) <- 1
L <- t(chol(P)) # L: LL^T = P
Y <- t(L %*% Z) # (n,d)-matrix containing n d-vectors following N(0,P)
U. <- runif(n)
W <- sapply(nu, function(nu.) 1/qgamma(U., shape = nu./2, rate = nu./2)) # (n,d)-matrix
X <- sqrt(W) * Y # (n,d)-matrix
U <- sapply(1:d, function(j) pt(X[,j], df = nu[j])) # (n,d)-matrix

## 2) Plot the data with a pairs plot, colorizing the groups
```

```
cols <- matrix("black", nrow = d, ncol = d) # colors
start <- c(1, cumsum(head(d., n = -1))+1) # block start indices
end <- cumsum(d.) # block end indices
for(j in seq_along(d.)) cols[start[j]:end[j], start[j]:end[j]] <- basecol[j] # colors
diag(cols) <- NA # remove colors corresponding to diagonal entries
cols <- as.vector(cols) # convert to a vector
cols <- cols[!is.na(cols)] # remove NA entries corresponding to diagonal
count <- 0 # panel number
my_panel <- function(x, y, ...) # panel function for colorizing groups
    { count <<- count + 1; points(x, y, pch = ".", col = cols[count]) }
pairs(U, panel = my_panel, gap = 0,
      labels = as.expression( sapply(1:d, function(j) bquote(italic(U[.(j)]))) ))

## 3) Zenplot of a random path through all pairs, colorizing the respective group
## Define our own points_2d_grid() for colorizing the groups
my_points_2d_grid <- function(zargs, basecol, d.) {
    r <- extract_2d(zargs) # extract information from zargs
    x <- r$x
    y <- r$y
    xlim <- r$xlim
    ylim <- r$ylim
    num2d <- zargs$num/2
    vars <- as.numeric(r$vlabs[num2d:(num2d+1)]) # two variables to be plotted
    ## Alternatively, we could have used ord[r$vars[num2d:(num2d+1)]] with
    ## the order 'ord' (see below) being passed to my_points_2d_grid()
    col <- if(all(1 <= vars & vars <= d.[1])) { basecol[1] } else {
           if(all(d.[1]+1 <= vars & vars <= d.[1]+d.[2])) { basecol[2] } else {
               if(all(d.[1]+d.[2]+1 <= vars & vars <= d)) basecol[3] else "black"
           }
    } # determine the colors
    vp <- vport(zargs$ispace, xlim = xlim, ylim = ylim, x = x, y = y) # viewport
    pointsGrob(x = x, y = y, pch = 21, size = unit(0.02, units = "npc"),
               name = "points_2d", gp = gpar(col = col), vp = vp)
}
## Plot a random permutation of columns via a zenplot
## Note: We set column labels here, as otherwise the labels can only
##       show *indices* of the variables to be plotted, i.e., the column
##       number in U[,ord], and not the original column number in U (which
##       is what we want to see in order to see how our 'path' through
##       the pairs of variables looks like).
colnames(U) <- 1:d
set.seed(1)
(ord <- sample(1:d, size = d)) # path; 1:d would walk parallel to the secondary diagonal
zenplot(U[,ord], plot1d = "layout", plot2d = "layout", pkg = "grid") # layout
zenplot(U[,ord], # has correct variable names as column names
        pkg = "grid",
        plot1d = function(zargs) arrow_1d_grid(zargs, col = "grey50"),
        plot2d = function(zargs)
            gTree(children = gList(
                  my_points_2d_grid(zargs, basecol = basecol, d. = d.),
                  rect_2d_grid(zargs, width = 1.05, height = 1.05,
                               col = "grey50", lty = 3),
                  label_2d_grid(zargs, loc = c(1.06, -0.03),
                                just = c("left", "top"), rot = 90, cex = 0.45,
                                fontface = "bold") )))
## => The points are colorized correctly (compare with the pairs plot).



### Using ggplot2 ###############################################################

## Although not thoroughly tested, in principle ggplot2 can also be used via
## pkg = "grid" as follows.

library(ggplot2)

## Define our own 2d plot
my_points_2d_ggplot <- function(zargs, extract2d = TRUE)
{
    if(extract2d) {
        r <- extract_2d(zargs) # extract results from zargs
        df <- data.frame(r$x, r$y) # data frame
names(df) <- c("x", "y")
        cols <- zargs$x[,"Species"]
    } else {
        ii <- plot_indices(zargs) # the indices of the variables to be plotted
        irs <- zargs$x # iris data
        df <- data.frame(x = irs[,ii[1]], y = irs[,ii[2]]) # data frame
        cols <- irs[,"Species"]
    }
    num2d <- zargs$num/2 # plot number among all 2d plots
    p <- ggplot() + geom_point(data = df, aes(x = x, y = y, colour = cols),
                               show.legend = num2d == 3) +
                    labs(x = "", y = "") # 2d plot
    if(num2d == 3) p <- p + theme(legend.position = "bottom", # legend for last 2d plot
                                  legend.title = element_blank())
    ggplot_gtable(ggplot_build(p)) # 2d plot as grob
}

## Plotting
iris. <- iris
colnames(iris.) <- gsub("\\.", " ", x = colnames(iris)) # => nicer 1d labels
zenplot(iris., n2dplots = 3, plot2d = "my_points_2d_ggplot", pkg = "grid")
zenplot(iris., n2dplots = 3,
        plot2d = function(zargs) my_points_2d_ggplot(zargs, extract2d = FALSE),
        pkg = "grid")



### Providing your own data structure ##########################################


## Danger zone: An example with a new data structure (here: a list of *lists*)
## Note: - In this case, we most likely need to provide both plot1d and plot2d
##         (but not in this case here since arrow_1d_graphics() does not depend
##         on the data structure)
##       - Note that we still make use of zargs here.
##       - Also note that the variables are not correctly aligned anymore:
##         In the ggplot2 examples we guaranteed this by plot_indices(),
##         but here we don't. This then still produces our layout but the
##         x/y axis of adjacent plots might not be the same anymore. This is
##         fine if only a certain order of the plots is of interest, but
##         not a comparison between adjacent plots.
z <- list(list(1:5, 2:1, 1:3), list(1:5, 1:2))
zenplot(z, n2dplots = 4, plot1d = "arrow", last1d = FALSE,
    plot2d = function(zargs, ...) {
       r <- unlist(zargs$x, recursive = FALSE)
       num2d <- zargs$num/2 # plot number among 2d plots
       x <- r[[num2d]]
       y <- r[[num2d + 1]]
       if(length(x) < length(y)) x <- rep(x, length.out = length(y))
       else if(length(y) < length(x)) y <- rep(y, length.out = length(x))
       plot(x, y, type = "b", xlab = "", ylab = "")
}, ispace = c(0.2, 0.2, 0.1, 0.1))



### Zenplots based on 3d lattice plots #########################################


library(lattice)
library(grid)
library(gridExtra)

## Build a list of cloud() plots (trellis objects)
## Note:
## - 'grid' problem: Without print(), the below zenplot() may fail (e.g.,
##   in fresh R sessions) with: 'Error in UseMethod("depth") :
##   no applicable method for 'depth' applied to an object of class "NULL"'
## - col = "black" inside scales is needed to make the ticks show
mycloud <- function(x, num) {
    lim <- extendrange(0:1, f = 0.04)
    print(cloud(x[, 3] ~ x[, 1] * x[, 2], xlim = lim, ylim = lim, zlim = lim,
                xlab = substitute(U[i.], list(i. = num)),
                ylab = substitute(U[i.], list(i. = num + 1)),
                zlab = substitute(U[i.], list(i. = num + 2)),
                zoom = 1, scales = list(arrows = FALSE, col = "black"),
                col = "black",
                par.settings = list(standard.theme(color = FALSE),
                                    axis.line = list(col = "transparent"),
                                    clip = list(panel = "off"))))
}
plst.3d <- lapply(1:4, function(i)
    mycloud(x[,i:(i+2)], num = i)) # list of trellis objects

## Preparing the zenplot
num <- length(plst.3d)
ncols <- 2
turns <- c(rep("r", 2*(ncols-1)), "d", "d",
           rep("l", 2*(ncols-1)), "d")
plot2d <- function(zargs) {
    num2d <- (zargs$num+1)/2
    vp <- vport(zargs$ispace, xlim = 0:1, ylim = 0:1)
    grob(p = zargs$x[[num2d]], vp = vp, cl = "lattice") # convert trellis to grid object
    ## Note: For further plots, Work with
    ##       gTree(children = gList(grob(zargs$x[[num2d]], vp = vp,
    ##                                   cl = "lattice")))
}

## Zenplot
## Note: We use a list of *plots* here already (not data)
zenplot(plst.3d, turns = turns, n2dplots = num, pkg = "grid", first1d = FALSE,
        last1d = FALSE, plot1d = "arrow_1d_grid", plot2d = plot2d)
```

Index