Package: graphicsutils (via r-universe)

September 8, 2024

Type Package

Title Collection of graphics utilities

Version 1.6.0.9000

Date 2022-01-08

Description A collection of functions to easily customize graphics-based plots.

Depends R (>= 3.2.0)

Imports grDevices, graphics, methods, stats, utils, jpeg (>= 0.1-8), png (>= 0.1-7), Rcpp (>= 0.12.9)

Suggests knitr, plotrix, rmarkdown, testthat, maps

LinkingTo Rcpp

NeedsCompilation yes

LazyData true

URL https://github.com/inSileco/graphicsutils, http://insileco.github.io/graphicsutils/

BugReports https://github.com/inSileco/graphicsutils/issues

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.1

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Repository https://insileco.r-universe.dev

RemoteUrl https://github.com/inSileco/graphicsutils

RemoteRef HEAD

RemoteSha 5c07f2b54493ac78d75c610020aabf759c50c89f

Contents

addAxis . . . 3
addFrame . . . 4
addGrid . . . 5
arrows2 . . . 6
biBoxplot . . . 8
blendColors . . . 9
box2 . . . 9
boxplot2 . . . 10
circles . . . 12
colorScale . . . 13
compassRose . . . 14
contrastColors . . . 16
dfGantt . . . 17
donut . . . 17
ellipse . . . 19
encircle . . . 20
envelop . . . 21
frameIt . . . 22
ganttChart . . . 23
getAngle2d . . . 24
gpuPalette . . . 25
gpuPalettes . . . 25
graphicsutils . . . 26
homothety . . . 26
howManyRC . . . 27
image2 . . . 27
interactiveLayout . . . 29
pchImage . . . 30
percX . . . 31
pickColors . . . 32
plot0 . . . 32
plotAreaColor . . . 33
plotImage . . . 34
plotMeans . . . 35
plotOnSide . . . 36
pointsInPolygon . . . 37
polarPlot . . . 38
prettyRange . . . 39
ramp . . . 40
rhombi . . . 41
rotation . . . 42
shadowText . . . 43
showPalette . . . 44
stackedAreas . . . 45
textBox . . . 46
toFig . . . 50

translation 51
vecfield2d 52

Index **54**

`addAxis` *Add Spatial Cardinal Directions to Axis Labels*

Description

This function adds cardinal directions to axis labels in a map.

Usage

```
addAxis(side, at = axTicks(side), mgp = par()$mgp, digits = 0, ...)
```

Arguments

`side`	an integer between 1 (bottom x-Axis) and 4 (right y-Axis).
`at`	values on the axis to add labels.
`mgp`	customize axis positions. See `par()`.
`digits`	number of digits of labels.
`...`	other parameters to pass to `axis()`.

Author(s)

Nicolas CASAJUS, <nicolas.casajus@gmail.com>

Examples

```
maps::map()
addFrame(grid = TRUE, add = TRUE, width = 5)
addAxis(1)
addAxis(2, at = seq(-50, 50, 100), digits = 2)
addAxis(3, lwd = 0, lwd.ticks = 0.5)
addAxis(4, las = 1, col.axis = "red")
```

addFrame *Add a Checkerboard-Like Frame around a Plot*

Description

This function adds a checkerboard-like frame around an existing plot (or a new one) as an ensemble of small rectangles whose colors, position and width can be specified by user.

Usage

```
addFrame(
  at_x,
  at_y,
  col = c("white", border),
  border = "black",
  width = NULL,
  lwd = 1,
  lty = 1,
  asp = NA,
  grid = FALSE,
  add = FALSE,
  ...
)
```

Arguments

at_x	positions on the x-Axis of the borders of rectangles.
at_y	positions on the y-Axis of the borders of rectangles.
col	one or several colors of rectangles background.
border	color of rectangles border.
width	width of the frame.
lwd	thickness of rectangles border (and grid).
lty	type of rectangles border lines (and grid).
asp	aspect ratio of the plot (see plot.window()).
grid	a boolean. If TRUE adds vertical and horizontal lines at at_x and at_y.
add	a boolean. If TRUE adds the frame to an existing plot.
...	other graphical parameters as in par().

Details

If add = FALSE, a new plot is drawn with xlim = ylim = c(-1, 1). Argument col can be set with one or multiple colors. If only one color is specified in col, border color will be used to draw the checkerboard. If width is NULL, the width of the frame is equivalent to 4% of the extent of the plot area.

Author(s)

Nicolas Casajus, <nicolas.casajus@gmail.com>

Examples

```
## Grid and graduations
addFrame()
addFrame(grid = TRUE)
addFrame(grid = TRUE, lwd = 0.25, lty = 3)
addFrame(at_x = seq(-1, 1, by = 0.1))

## Frame width
addFrame(width = 0.25)
addFrame(at_x = seq(-1, 1, by = 0.01), width = 1.00)

## Colors
addFrame(col = 1:6)
addFrame(col = "lightgray", border = "darkgray")

## Adding to a plot
maps::map()
addFrame(grid = TRUE, add = TRUE, width = 5)
maps::map.axes()

## Other graphical parameters
addFrame(bg = "darkgray", xaxs = "i", yaxs = "i")
```

---

addGrid *Add a Grid to a Plot*

Description

This function adds a grid to an existing plot (like box()). User can specify coordinates on x- and/or y-axis where to draw vertical and/or horizontal lines respectively. User can also color the background of the plot area and add a box around this area (if required).

Usage

```
addGrid(
  at_x,
  at_y,
  col = NA,
  border = "black",
  lwd = 1,
  lty = 1,
  box = FALSE,
  ...
)
```

Arguments

at_x, at_y	coordinates on the x and y-axis where to draw lines.
col	the color of the background.
border	the color of lines (and box).
lwd	the width of lines (see par()).
lty	the type of lines (see par()).
box	a boolean. If TRUE add a box around the plot area.
...	other graphical parameters as in par().

Details

If user does not specify at_x and at_y, the grid is aligned with the tick marks on the corresponding default axes (computed by axTicks).

Author(s)

Nicolas CASAJUS, <nicolas.casajus@gmail.com>

Examples

```
maps::map()
addGrid()
addGrid(box = TRUE)
addGrid(col = "#ff000044", border = "green", box = TRUE)

maps::map()
addGrid(at_x = axTicks(1), border = "blue", lwd = 2)
addGrid(at_y = axTicks(2), border = "red")
addGrid(at_x = NULL, at_y = NULL, lwd = 3, box = TRUE)
```

---

arrows2 *Add arrows to a plot*

Description

Draw a custom arrows between pairs of points.

Usage

```
arrows2(
  x0,
  y0,
  x1 = x0,
  y1 = y0,
  off0 = 0,
```

```
  off1 = off0,
  cex.arr = 1,
  cex.shr = 1,
  cex.hh = 1,
  cex.hl = 1,
  prophead = TRUE,
  twoheaded = FALSE,
  ...
)
```

Arguments

x0	the x coordinates of points from which to draw arrows.
y0	the y coordinates of points from which to draw arrows.
x1	the x coordinates of points to which to draw arrows.
y1	the y coordinates of points to which to draw arrows.
off0	offset of points from which to draw arrows.
off1	offset of points to which to draw arrows.
cex.arr	the magnification coefficient to be used for the heights of the arrows.
cex.shr	the magnification coefficient to be used to change the height of arrows towards their heads.
cex.hh	the magnification coefficient to be used for the heights of arrows' head.
cex.hl	the magnification coefficient to be used for the lengths of arrows' head.
prophead	logical. If TRUE arrows are drawn with head proportional to the length of the arrows.
twoheaded	logical. If TRUE two-headed arrows are drawn, default is FALSE.
...	additional arguments to be passed to polygon function.

See Also

graphics::arrows()

Examples

```
# Example 1:
plot0(c(0, 10), c(0, 10))
arrows2(1, 9, 8)
arrows2(1, 8, 8, 1, cex.hh = 1.2, cex.hl = 1.2, col = "grey30", lwd = 1.2,
 prophead = TRUE, twoheaded = TRUE)
arrows2(5, 9, 5, 1)

# Example 2:
plot0(c(0, 1), c(0, 1))
arrows2(runif(2), runif(2), x1 = runif(2), y1 = runif(2))
arrows2(runif(2), runif(2), x1 = runif(2), y1 = runif(2), prophead = FALSE,
 lty = 3)
```

`biBoxplot` *A bi-boxplot*

Description

Draws boxplots and bi-boxplots.

Usage

```
biBoxplot(
  df1,
  df2 = df1,
  probs = c(0.01, 0.25, 0.5, 0.75, 0.99),
  width = 0.2,
  sta_wd = 0.5,
  median = NULL,
  staples = NULL,
  whiskers = NULL,
  col_left = "grey75",
  col_right = col_left,
  add = FALSE,
  at = NULL
)
```

Arguments

`df1`, `df2`	first and second set of boxplots.
`probs`	numeric vector of five probabilities, see `stats::quantile()`.
`width`	a vector giving the relative widths of the boxes making up the plot.
`sta_wd`	staple width.
`median`	a list of arguments passed to `graphics::lines()` to custom the median line.
`staples`	a list of arguments passed to `graphics::lines()` to custom the staples.
`whiskers`	a list of arguments passed to `graphics::lines()` to custom the whiskers.
`col_left`	color of the left boxes.
`col_right`	color of the right boxes.
`add`	a logical. Should the biboxplots be added on the current graph? If `FALSE` then a new plot is created.
`at`	numeric vector giving the locations where the boxplots should be drawn. Same default behavior as in `graphics::boxplot()`.

Details

This functions does not attempt to assess the distributions as it is based on quantiles.

Examples

```
dis1 <- list(stats::rnorm(1000, sd = 0.25))
biBoxplot(dis1)
```

`blendColors` *Blend colors*

Description

Blend a set of colors and return the color thereby obtained.

Usage

```
blendColors(cols)
```

Arguments

`cols`	set of colors to be mixed.

Examples

```
blendColors(c("blue", "purple"))
```

`box2` *Alternative box function*

Description

Draw a box around a plot.

Usage

```
box2(side, which = c("plot", "figure", "outer", "inner"), fill = NULL, ...)
```

Arguments

`side`	a numerical or character vector or a character string specifying which side(s) of the plot the box is to be drawn (see details).
`which`	a character, one of `plot`, `figure`, `inner` and `outer`.
`fill`	the color to be used to fill the box.
`...`	further graphical parameters (see `graphics::par()`) may also be supplied as arguments, particularly, line type, `lty`, line width, `lwd`, color, `col` and for `type = 'b'`, `pch`. Also the line characteristics `lend`, `ljoin` and `lmitre`.

Details

This function intends to give more flexibility to the graphics::box() function. As which parameter, the user provides an object first coerced by as.character to a character string that is secondly split into single characters. For all of these characters, matches are sought with all elements of 1, 2, 3, 4, b, l, t, r where 1=below, 2=left, 3=above, 4=right, b=below, l=left, t=above and r=right.

See Also

graphics::box()

Examples

```
# Example 1:
plot0()
box2()

# Example 2:
plot0()
box2("14", fill = "grey80", lwd = 2)
plot0()
box2(c(1, 4), fill = "grey80", lwd = 2)
plot0()
box2(c(1, 4), fill = "grey80", lwd = 2)

# Example 3:
par(mfrow = c(2, 2), oma = c(2, 2, 2, 2))
plot0(0, 0)
box("outer", lwd = 2)
box("inner", lwd = 2)
plot.default(0, 0)
plot.default(0, 0)
plot0()
box2(which = "figure", lwd = 2, fill = 2)
box2(side = 12, lwd = 2, fill = 8)
```

---

boxplot2 *Custom boxplots*

Description

Produce custom box-and-whisker plot(s) of the given (grouped) values.

Usage

```
boxplot2(
  x,
  ...,
  probs = c(0.05, 0.25, 0.5, 0.75, 0.95),
```

```
  vc_cex = c(5, 18, 1.4),
  colors = c("grey75", "grey75", "grey20"),
  med_pch = 3,
  add = FALSE,
  at = NULL
)
```

Arguments

x	a formula, a an object to be coerced as a data frame.
...	further arguments to be passed to `stats::aggregate()`.
probs	a numeric vector of 5 probabilities used to drawn the boxes. By default `probs=c(.05, 0.25, .5, .75, .95)`.
vc_cex	numeric vector of 3 magnifications coefficients. The first one determines the width of the whiskers, the second one is for the width of the box and the last one is for the symbol indicating the median.
colors	numeric vector of 3 colors for 1- the whiskers 2- the boxes and 3- the symbols that indicates the median.
med_pch	pch value used to indicate the median.
add	a logical. Should the boxes be added on the current graph?
at	numeric vector giving the locations where the boxplots should be drawn, particularly when 'add = TRUE'; defaults to '1:n' where 'n' is the number of boxes.

Value

Draw a boxplot and returns the coordinates as an invisible output.

Author(s)

Kevin Cazelles

Examples

```
boxplot2(replicate(10, runif(100)))
dfa <- data.frame(name = rep(LETTERS[1:4], 25), val = runif(100))
plot0(c(0, 5), c(0, 1))
boxplot2(val ~ name, data = dfa, add = TRUE, vc_cex = c(4, 40, 2))
```

`circles` *Draw circles*

Description

Draw circles in a flexible way.

Usage

```
circles(
  x,
  y = x,
  radi = 1,
  from = 0,
  to = 2 * pi,
  incr = 0.01,
  pie = FALSE,
  clockwise = FALSE,
  add = TRUE,
  ...
)
```

Arguments

`x, y`	the x and y coordinates of the centers of the circles.
`radi`	the radii of the circles.
`from`	the angles, expressed in radians, from which circles are drawn.
`to`	the angles, expressed in radians, to which circles are drawn.
`incr`	increments between two points to be linked (expressed in radians).
`pie`	a logical. If `TRUE`, end points are linked with the center of the circle (default is set to `FALSE`).
`clockwise`	a logical. Shall circles and arcs be drawn clockwise? Defaut is `FALSE`.
`add`	a logical. Should the circles be added on the current plot?
`...`	additional arguments to be passed to `graphics::polygon()` function.

Details

The number of circles drawn is given by the maximum argument length amng `x`, `y`, `radi`, `from` and `to` arguments. Sizes are adjusted (i.e. repeated over) with `rep_len()` function.

Value

An invisible list of `data.frame` of two columns including the coordinates of all circles.

See Also

graphics::symbols(), plotrix::draw.circle(), plotrix::draw.arc().

Examples

```
# Example 1:
plot0()
circles(x = 0)

# Example 2:
plot0()
circles(x = -.5, radi = 0.45, from = 0.5 * pi, to = 0.25 * pi)
circles(x = .5, radi = 0.45, from = 0.5 * pi, to = 0.25 * pi, pie = TRUE)
circles(x = .5, y = -.5, radi = 0.45, from = 0.5 * pi, to = 0.25 * pi, pie
= TRUE, clockwise = TRUE)

# Example 3:
plot0()
circles(matrix(-.5 + .5 * stats::runif(18), ncol = 3))

# Example 4:
plot0(x = c(-2, 2), y = c(-2, 2), asp = 1)
circles(x = c(-1, 1), c(1, 1, -1, -1), from = pi * seq(0.25, 1, by = 0.25),
to = 1.25 * pi, col = 2, border = 4, lwd = 3)
```

---

colorScale *Create a color scale*

Description

Simple and flexible color scale.

Usage

```
colorScale(
  x,
  y,
  col,
  at = NULL,
  labels = NULL,
  horiz = TRUE,
  percx = NULL,
  percy = NULL,
  adj = 0,
  labels.cex = 1,
  title = "legend",
  title.cex = 1.2
)
```

Arguments

x, y	x and y coordinates of the bottom-left corner of the legend.
col	vector of colors.
at	vector of integers indicating colors to be labeled.
labels	labels.
horiz	a logical. Should the color scale be horizontal?
percx, percy	size of the color scale along the x-axis and the y-axis.
adj	adjust the position labels position. If `adj = 0` then labels are at the bottom if `horiz = TRUE` and on the the left if `horiz = FALSE`. If `adj = 1`, labels are at the top if `horiz = TRUE` and on the the right if `horiz = FALSE`. So far if `horiz = TRUE`, `title` and `labels` are on opposite sides.
labels.cex	magnification to be used for labels.
title	legend title.
title.cex	magnification to be used for the title.

Examples

```
plot(c(0, 10), c(0, 10))
colorScale(5, 5, gpuPalette("cisl", 10), at = c(2, 4))
```

---

compassRose *Compass Rose*

Description

This function draws a fully personnalisable compass rose.

Usage

```
compassRose(
  x = 0,
  y = 0,
  labels = c("S", "W", "N", "E"),
  rot = 0,
  cex.cr = 1,
  cex.let = cex.cr,
  col.cr = c(1, 8),
  col.let = 1,
  border = c(1, 8),
  offset = 1.2,
  add = TRUE,
  ...
)
```

```
compassRoseCardinal(
  x,
  y = x,
  rot = 0,
  cex.cr = 1,
  cex.let = 1,
  labels = c("S", "W", "N", "E"),
  offset = 1.2,
  col.cr = c(1, 8),
  col.let = 1,
  border = c(1, 8),
  ...
)
```

Arguments

x, y	the x and y coordinates of the center of the compass rose.
labels	a vector of four character strings used as labels for the cardinal directions.
rot	rotation for the compass rose in degrees (clockwise).
cex.cr	the magnification to be used for the whole compass rose.
cex.let	the magnification to be used for labels.
col.cr	a vector of colors used to draw compass rose (see details).
col.let	a character string specifying the labels' color.
border	a vector of colors of the borders of the compass rose.
offset	label offset of the cardinal points.
add	a logical. Should the compass rose be added on the current graph?
...	additional arguments to be passed to graphics::polygon().

Details

Both col.cr and border are repeated over (base::rep() is called) so it has a 8 elements, meaning all triangles the compass rose is made of could have their own color.

Note that there already exists a similar function by Jim Lemon in sp package.

Functions

- compassRose(): A compass rose with the four cardinal directions and additional directions.
- compassRoseCardinal(): A compass with the four cardinal directions only.

Examples

```
compassRose(0, rot = 25, cex.cr = 2, col.let = 2, add = FALSE)
```

`contrastColors` *Contrast colors.*

Description

For a given set of colors, `contrastColors` returns an associated set of colors.

Usage

```
contrastColors(colors, how = "how_borw", alpha = FALSE)

col2Hex(colors, alpha = FALSE)
```

Arguments

`colors`	vector of any of the three kinds of R color specifications, see `grDevices::col2rgb()`.
`how`	a method to contrast `colors`. Methods currently available are `how_borw`, `how_cent`, `how_oppo` and `how_prop`, see details.
`alpha`	logical value indicating whether the alpha channel (opacity) values should be returned.

Details

Based on the sum of colors' saturation `how_borw` returns black or white, `how_prop` proportionally remove or add some saturation. `how_oppo` opposes the color (255-x) and `how_cent` centers the columns, i.e. remove or add 127.

Functions

- `contrastColors()`: Retuns a set of colors contrasted.
- `col2Hex()`: Returns the hexadecimal string associates to a given vector of colors.

See Also

`grDevices::col2rgb()`

Examples

```
contrastColors("blue")
contrastColors("blue", how = "how_prop")
```

dfGantt *dfGantt*

Description

A data frame to showcase `ganttChart()`.

Usage

```
data(dfGantt)
```

Format

A simulated data frame to create a Gantt chart.

Author(s)

David Beauchesne

---

donut *Donut chart*

Description

Draw a donut chart.

Usage

```
donut(
  vec,
  eaten = 0,
  labels = NULL,
  rot = 0,
  cex = 0.8,
  tck = 0.05,
  width = 0.6,
  mycol = 1 + seq_along(vec),
  density = NULL,
  angle = 45,
  dt = 0.001,
  add = FALSE,
  cx = 0,
  cy = 0,
  border = NA,
  clockwise = TRUE,
  ...
)
```

Arguments

vec	a vector of non-negative numerical quantities that are displayed as the areas of donut slices.
eaten	the eaten part.
labels	one or more expressions or character strings giving names for the slices.
rot	the rotation angle (in degree).
cex	the magnification coefficient to be used for the size of the donut.
tck	the magnification coefficient to be used for the length of the tick marks.
width	the width of the donut (set between 0 and 1).
mycol	vector of colors to be used.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise).
dt	point density of the drawn circles.
add	a logical. Should the donut chart be added on the current graph? If FALSE then a new plot is created.
cx	the magnification coefficient to be used for the total horizontal width of the donut.
cy	controls the total vertical width of the donut.
border	the border color of the donut, set to NA which omits borders.
clockwise	a logical. Shall slices be drawn clockwise?
...	additional arguments to be passed to lines methods.

Details

As pie chart, donut charts are a very bad way of displaying information, see graphics::pie() The aspect of the donut is fully customizable. If width is set to 1 and eaten to 0, the donut chart is then a pie chart.

Note

Substantial part of the code have been inspired by the pie function.

The 'col' argument determines the succession of colors to be applied to each axis.

Examples

```
# Example 1:
graphics::par(mfrow = c(2, 2), mar = rep(2, 4))
donut(vec = c(10, 20, 15))
donut(c(10, 20, 15), eaten = 0.2)
donut(c(10, 20, 15), eaten = 0.2, rot = 180, labels = paste("group", 1:3), lwd = 3, col = 8)
donut(c(10, 20, 15), 0.2, cx = 4, col = 4, mycol = c(4, 3, 2), density = 30, angle = c(20, 55, 110))

# Example 2:
```

```
plot0(c(0, 10), c(0, 40), type = "n")
vec <- runif(7)
donut(vec, 0.15, cx = 5, cy = 20, add = TRUE, col = 2)
```

`ellipse` *Draw ellipses*

Description

Draw ellipses in a flexible way.

Usage

```
ellipse(
  x = 0,
  y = x,
  mjradi = 1,
  mnradi = 0.5,
  from = 0,
  to = 2 * pi,
  rot = 0,
  incr = 0.01,
  pie = FALSE,
  ...
)
```

Arguments

`x`	the x coordinates of the centers of the ellipses.
`y`	same as `x` for the y-axis.
`mjradi`	the major radii of the ellipses.
`mnradi`	the minor radii of the ellipses.
`from`	the angles, expressed in radians, from which ellipses are drawn.
`to`	the angles, expressed in radians, to which ellipses are drawn.
`rot`	the rotation angles (in degree) of the ellipses.
`incr`	increments between two points to be linked (expressed in radians).
`pie`	a logical. If `TRUE` end points are linked with the center of the ellipse (default is set to `FALSE`).
`...`	additional arguments to be passed to `graphics::polygon()`.

Details

For a rotation angle of 0, major radii refer to the one along the x axis. The number of circles drawn is given by the maximum argument length among `x`, `y`, `radi`, `from` and `to` arguments. Note vector length are adjusted using `rep_len()`.

Note

There is a similar function, called `draw.ellipse`, in the package `plotrix`.

Examples

```
# Example 1:
plot0(asp = 1)
ellipse()

# Example 2:
plot0()
for (i in seq(0, 360, 30)) ellipse(rot = i)

# Example 3:
plot0()
ellipse(matrix(-.5 + .5 * stats::runif(18), ncol = 3))

# Example 4:
plot0(x = c(-2, 2), y = c(-2, 2), asp = 1)
ellipse(
    x = c(-1, 1), c(1, 1, -1, -1), from = pi * seq(0.25, 1, by = 0.25),
    to = 1.25 * pi, col = 2, border = 4, lwd = 3
)
```

`encircle` *Encircle points*

Description

Draw a polygons around a certains set of points.

Usage

```
encircle(x, y = NULL, nb.pt = 20, off.set = 1, ...)
```

Arguments

`x`	the x coordinates of a set of points. Alternatively, a single argument `x` can be provided.
`y`	the y coordinates of a set of points.
`nb.pt`	the number of points to be generated around each coordinates.
`off.set`	the y coordinates of a set of points.
`...`	further arguments to be passed to `graphics::polygon()` function.

Details

The technique employed is straightforward: for each point, a circle of radius of off.set`made of`nb.pt‘ points, then a convex is drawn around the coordinates using `grDevices::chull()`.

Examples

```
coords <- cbind(runif(10), runif(10))
plot0(coords)
points(coords, bg='grey25', pch=21)
encircle(coords, border='#7b11a1', lwd=2)
```

---

envelop *Compute the coordinates of an envelop*

Description

envelop eases the computation of the polygons described by to set of y coordinates along the x-axis. Optionally, the polygons could be added on the current plot.

Usage

```
envelop(x, upper, lower = rep(0, length(upper)), add = TRUE, ...)
```

Arguments

x	vectors containing the x coordinates.
upper	the y coordinates of the upper values.
lower	the y coordinates of the lower values.
add	a logical. If TRUE the envelop is drawn as a polygon (default behavior).
...	additional arguments to be passed to graphics::polygon() function.

Value

The coordinates of the envelop are returned if assigned.

Examples

```
plot0(c(0, 10), c(0, 10))
sz <- 100
seqx <- seq(0, 10, length.out = sz)
seqy1 <- 0.2 * seqx * runif(sz, 0, 1)
seqy2 <- 4 + 0.25 * seqx * runif(sz, 0, 1)
seqy3 <- 8 + 0.25 * seqx * runif(sz, 0, 1)
envelop(seqx, seqy1, seqy2, col = 'grey85', border = NA)
envelop(seqx, seqy2, seqy3, col = 'grey25', border = NA)
```

`frameIt` *Draw a frame around a plot*

Description

Draw a frame around the plot region as an ensemble of small rectangles whose colors can be specified.

Usage

```
frameIt(
  nbc = 10,
  cex.x = 1,
  cex.y = cex.x,
  col = c("grey45", "grey85"),
  border = NA
)
```

Arguments

`nbc`	number of rectangles to be drawn for each axis.
`cex.x`	control the dimension
`cex.y`	The radii of the circles.
`col`	color of the rectangles drawn.
`border`	color of the borders if the rectangles drawn.

Details

The number of rectangles could be different from `nbc` as `pretty` is called to properly locate them. The `col` argument determines the succession of colors to be applied to each axis.

Examples

```
# Example 1:
plot0()
frameIt()

# Example 2:
plot0()
frameIt(cex.x = 1.5, col = c(2, 3), border = 1)
```

ganttChart *Gantt Chart*

Description

A flexible Gantt Chart.

Usage

```
ganttChart(
  df,
  task_order = TRUE,
  mstone_add = task_order,
  mstone_spacing = 1,
  mstone_lwd = 2,
  axes = TRUE,
  mstone_font = 2,
  lighten_done = 0
)
```

Arguments

df	a data.frame (see details).
task_order	a logical. Should the tasks be ordered? See below for more details.
mstone_add	Should milestones be added?
mstone_spacing	spacing between milestones (expressed as figure unit).
mstone_lwd	lines width for the milestone.
axes	a logical. Should the axes be added?
mstone_font	font of milestone (ignore of axes is FALSE).
lighten_done	percentage use to lighten done task (see lighten()). Default set to 0, so the completed task of a given milestone have the same color as the pending ones.

Details

Argument df should be a data frame with the following columns (in any order):

- milestone: milestones names,
- due: due date (will be converted into a date with as.Date()),
- start: start date (will be converted into a date with as.Date()),
- task: tasks names. It might as well include any of the following optional columns:
- done: vector of logicals indicating whether the task if completed
- col: to custom the color of the tasks.

Author(s)

David Beauchesne, Kevin Cazelles

References

https://insileco.github.io/2017/09/20/gantt-charts-in-r/

Examples

```
ff <- ganttChart(dfGantt,
  mstone_lwd = 3, mstone_spacing = 0.6,
  lighten_done = 80
)
```

---

getAngle2d *Angles between vectors.*

Description

Calcul the angles of two set of vectors.

Usage

```
getAngle2d(x0, y0, x1, y1, rad = FALSE)
```

Arguments

x0, y0	the x and y coordinates of the first set of vector.
x1, y1	the x and y coordinates of the second set of vector.
rad	a logical. Should angles be expressed radians?

Examples

```
# Example:
plot0(c(-10, 10), c(-10, 10))
arrows(c(0, 3), c(0, 0), c(3, 0), c(3, -3))
cool <- getAngle2d(c(0, 3), c(0, 0), c(3, 0), c(3, -3))
```

gpuPalette *graphicsutils colors palettes*

Description

A couple of color palettes including our inSileco palette and a couple of other inspired by popular websites.

Usage

```
gpuPalette(id, ncolors)
```

Arguments

id	either an integer or a character string matching the names of the desired color palette(s). If missing then color palettes available are prompted.
ncolors	An integer giving the number of colors to be returned. If greater than the number of colors included in the selection, then 'grDevices::colorRampPalette() is called to expand the palette. If missing, then the entire palette is returned.

Value

A vector of character strings of the hexadecimal colors values. Note that if several color palette is requested, then the output is a vector with all color palette concatenated.

Examples

```
showPalette(gpuPalette(c('atom', 'cisl')))
showPalette(gpuPalette('cisl', 100))
```

gpuPalettes *Complete list of graphicsutils' color palettes*

Description

Color palettes of the graphicsutils package.

Usage

```
gpuPalettes
```

Format

An object of class `list` of length 7.

graphicsutils *graphicsutils*

Description

A collection of (hopefully) useful functions to create graphics based plots.

homothety *Homothety*

Description

Compute a homothetic transformation for a set of points. The transformed set of points is optionally drawn as a polygon.

Usage

```
homothety(x, y, lambda, xcen = NULL, ycen = NULL, add = FALSE, ...)
```

Arguments

x, y	the x and ycoordinates of points. It can also be a matrix (see details).
lambda	the factor to be used for the homothetetic transformations.
xcen, ycen	the x and y coordinate for the center of rotation.
add	logical. If TRUE the set of transformed points are drawn as a polygon.
...	additional arguments to be passed to graphics::polygon() function (used only if add is TRUE).

Details

If x is a matrix with more than 2 columns, then x is the first column and y the second one.

Note that lambda, xcen and ycen are unique, meaning that homothety computes only one homothetic transformation. Drawing the points computed is relevant only if there are more than 2 points.

References

https://en.wikipedia.org/wiki/Homothetic_transformation.

Examples

```
# Example:
plot0(c(0, 10), c(0, 10))
x <- c(4, 6, 5)
y <- c(2, 2, 4)
polygon(x, y)
poly2 <- homothety(x, y, 2)
polygon(poly2$x, poly2$y)
poly3 <- homothety(x, y, -2.5, xcen = 5, ycen = 4, border = 4, add = TRUE)
```

---

howManyRC *How many rows and columns*

Description

Compute the number of rows and columns to split a graphic window into panels as equally as possible.

Usage

```
howManyRC(n)
```

Arguments

n	a positive integer (or coercible as is).

Value

A vector of two elements: the number of rows followed by the number of columns.

---

image2 *Alternative image function*

Description

Creates a grid of colored or gray-scale rectangles. This function is similar to graphics::image() when used for a matrix but simpler (*i.e* less available features).

Usage

```
image2(
  x,
  from = NULL,
  to = NULL,
  color_scale = NULL,
  border = NA,
  add_value = FALSE,
  val_cex = 1,
  n_signif = 2,
  ...
)
```

Arguments

x	a matrix or an object to be coerced as a matrix.
from	matrix values equal to or smaller than from will be associated with the first color of the color scale.
to	values equal to or larger than from will be associated with the last color of the color scale.
color_scale	a vector of colors.
border	color for rectangle borders (see graphics::rect()).
add_value	a logical should value be added in the middle of the rectangles drawn?
val_cex	coefficient of magnification used if values are displayed.
n_signif	number of significant numbers to be displayed (used when labelc is NULL).
...	further arguments to be passed to graphics::rect().

Details

This function actually draws rectangles to create an image from a matrix. Unlike graphics::image(), image2 the image is ordered just as the matrix is displayed meaning that the cell (1,1) is at the upper left cell of the plot drawn. Note that currently neither titles nor axes' labels are added user should call the graphics::title() and graphics::axis(). Concerning the latter, the user should be aware that cell's coordinates range from 0 to 1 with 0 being the coordinates of the first cell and 1 the coordinates of the last cell (if there is only one cell then the center of the unique cell is 0).

See Also

graphics::image() graphics::rect()

Examples

```
image2(matrix(1:9, 3))
image2(matrix(1:9, 3), add_value = TRUE)
image2(matrix(1:27, 3), from=2, border = 2, lwd=2)
```

`interactiveLayout` *An interactive version of* `layout`

Description

`layout2()` provides an interactive version of `graphics::layout()`. Once called, an interactive grid is displayed and can be used to create a layout.

Usage

```
interactiveLayout(
  n = 1,
  grain.x = 20,
  grain.y = grain.x,
  show = TRUE,
  now = TRUE
)
```

Arguments

`n`	Number of plot regions, default is set to 1.
`grain.x`	Number of vertical lines drawn to select the size of the subplots regions.
`grain.y`	Number of horizontal lines drawn to select the size of the subplots regions.
`show`	logical. If `TRUE` `graphics::layout.show()` is used to get a preview of the subplots regions.
`now`	logical. If `TRUE` `graphics::layout()` is called on exit.

Details

Arguments `grain.x` and `grain.y` control the aspect of the support grid generated to locate the different panels. Once the grid pops up, the user must click 2*n times to select the size of the n subplots. A panel is delimited by two consecutive clicks, the tow cells selected by click* 2p and clicks 2*p+1 will be used to compute the area allocated to panel p. Note that the user is allowed to use the same area for several plots but only this area will actually be used only by the last one.

As `graphics::layout()` is ultimately called, `layout2()` has the same limits, i.e. currently 200 for the numbers of rows and columns and 10007 for the total number of cells.

Value

the matrix use to draw the layout is returned as an invisible output.

pchImage *Images as plotting characters*

Description

pchimage returns a plot that displays an image. It allows users to directly include a .png or a .jpeg file in a plot region.

Usage

```
pchImage(
  x,
  y,
  obj = NULL,
  file = NULL,
  cex.x = 1,
  cex.y = cex.x,
  atcenter = TRUE,
  add = TRUE,
  col = NULL,
  ...
)
```

Arguments

x	the x coordinates of images to be drawn.
y	the y coordinates of images to be drawn.
obj	an object of class nativeRaster.
file	a path to either a .png file or a .jpeg file.
cex.x	a numerical value giving the amount by which the horizontal width of the image should be magnified, a value of 1 means 5% of the total width.
cex.y	Same as cex.x for the y axis.
atcenter	a logical. If TRUE them x and y coordinates describe the center of the image. Otherwise they represent the bottom-left coordinates of the image.
add	logical. Should images be added on the current plot? If FALSE a new plot is created.
col	optional color use to fill pixels whose values are not 0.
...	Additional arguments to be passed to the rasterImage function.

Details

Either obj or file must be defined. If file is not null, then png::readPNG() or jpeg::readJPEG() according to the end of the file extension.

Examples

```
# Example:
img <- png::readPNG(system.file('img', 'Rlogo.png', package='png'), native=TRUE)
n<-15
plot0(c(0,1),c(0,1))
pchImage(0.1+0.8*stats::runif(n), 0.1+0.8*stats::runif(n), cex.x=0.2+1.6*stats::runif(n),
obj=img, angle=360*runif(n), col=2)
```

`percX` *Values at a given percentage of the axis*

Description

Returns the values of either x-axis or y-axis for a given percentage.

Usage

```
percX(percentage = 90)

percY(percentage = 90)
```

Arguments

`percentage`	The percentage of the axis for which the values is returned.

Details

This function intends to ease the positioning of additional marks on plot (such as annotation). Note that percentages refer to the entire plot window.

Functions

- `percX()`: Returns the values of x-axis for a given percentage.
- `percY()`: Returns the values of y-axis for a given percentage.

Examples

```
plot0()
text(x = percX(90), y = percY(90), label = "cool")
```

pickColors *Pick colors up*

Description

Generate an interactive interface to pick a set of colors up.

Usage

```
pickColors(n = 9, ramp = grDevices::rainbow(1024), nb_shades = 512)
```

Arguments

n	The number of colors to be selected (9 by default).
ramp	A vector of colors used as tone palette.
nb_shades	Number of shades to be displayed once a tone is selected.

Details

This function generates a graphical window split into 6 panels. The top panel serves to select one tone. The panel right below presents nb_shades of the selected tones. The bottom right panel displays the current selection that can be stored by clicking on the bottom left panel *Keep it*. The bottom center panel shows the characteristic of the selected color. Finally, in order to abort before completing the selection of colors, the user can simply click on the *Stop* panel (on the left).

Value

A character vector including the colors selected.

plot0 *An empty plot function*

Description

plot0 returns an a plot of a specific size without any symbols.

Usage

```
plot0(
  x = c(-1, 1),
  y = NULL,
  fill = NULL,
  text = NULL,
  grid.col = NULL,
  grid.lwd = 1,
  grid.lty = 1,
  ...
)
```

Arguments

x	the x coordinates of points in the plot or a matrix of coordinates.
y	the y coordinates of points in the plot.
fill	The color to be used to fill the plot area.
text	A character string or a object to be coerced as character string that will be displayed in the center of the plot region.
grid.col	color of the grid's lines. The default value is NULL, in which case the grid is not drawn.
grid.lwd	line width of the grid's lines.
grid.lty	line type of the grid's lines.
...	further graphical parameters from graphics::par() (such as srt) or graphics::plot.default.

Examples

```
# Example 1
plot0()

# Example 2
plot0(c(-10,10), asp=1)

# Example 3
plot0(c(-10,10), text='cool', cex = 2)

# Example 3
plot0(c(-10,10), asp=1, fill=8, text='cool', srt=45, cex=4, col=2)

# Example 4
plot0(c(-10,10), fill='#ebebeb', grid.col = 'white')

# Example 4
plot0(c(-10,10), grid.col = 2)
```

---

plotAreaColor *Color the plot area*

Description

Color the plot area by drawing a rectangle. Default color is 'grey80'.

Usage

```
plotAreaColor(color = "grey80", border = NA, ...)
```

Arguments

color	Color of the rectangle, default is set to 'grey80'.
border	Color of the border of the rectangle drawn, default is set to NA, that is no border.
...	Additional arguments to be passed to graphics::rect() function.

Details

The function calls graphics::rect() and draw a colored rectangle (default color is set to light blue) whose dimensions are given by argument usr of function graphics::par().

Note

In graphics::par(), argument bg colors the entire window.

Examples

```
#Example 1:
plot0()
plotAreaColor()

#Example 2:
plot0()
plotAreaColor(col=8, lwd=4, border=4)
```

---

plotImage *Display an image*

Description

Returns a plot that displays an image. It enables users to directly include a .png or a .jpeg file in a plot region by providing their path.

Usage

```
plotImage(obj = NULL, file = NULL, add = FALSE, ...)
```

Arguments

obj	an object of class nativeRaster function.
file	a path to either a .png file or a .jpeg file.
add	logical. Should images be added on the current graph? If FALSE a new plot is created.
...	additional arguments to be passed to rasterImage function.

Details

Note that either obj or file must be defined. If a path is provided either readPNG or readJPEG according to the end of the file extension.

Examples

```
img <- png::readPNG(system.file('img', 'Rlogo.png', package='png'),
native=TRUE)
op <- par(no.readonly = TRUE)
par(mfrow=c(4,4), mar=rep(2,4))
for (i in seq_len(16)) plotImage(img)
par(op)
```

`plotMeans` *Plots means and error associated*

Description

Plots a set of means computed based on a dataset and draw error associated.

Usage

```
plotMeans(
  formula,
  data,
  FUN_err = stats::sd,
  add = FALSE,
  seqx = NULL,
  draw_axis = TRUE,
  col_err = par()$col,
  col_pt = par()$col,
  cex_pt = 1,
  connect = FALSE,
  args_con = list(),
  ...
)
```

Arguments

`formula`	a formula, see `stats::formula()`.
`data`	a data frame (or list) from which the variables in formula should be taken.
`FUN_err`	the function that assess uncertainty. Default function is `stats::sd()`.
`add`	logical. should images be added on the current graph ? If `FALSE` a new plot is created.
`seqx`	the x coordinates of the means to be plotted, if `NULL`, default values are used. This is intended to be used when add parameter is `TRUE`.
`draw_axis`	logical. If `TRUE` axes and box are drawn.
`col_err`	color of the lines that reflect uncertainty.
`col_pt`	color of the points that stand for means.
`cex_pt`	magnification coefficient of the points that stand for means.

connect	logical. If TRUE then mean are linked using a lines.
args_con	a list of parameters that are used to customize the lines that links the means.
...	Further graphical parameters (see plot.default() and) may also be supplied as arguments, particularly, line type, lty, line width, lwd, color, col and for type = 'b', pch. Also the line characteristics lend, ljoin and lmitre.

Examples

```
# Example:
dataset <- data.frame(dat = c(rnorm(50, 10, 2), rnorm(50, 20, 2)),
 grp = rep(c("A", "D"), each = 50))
graphics::par(mfrow = c(1, 3))
plotMeans(dat ~ grp, data = dataset, pch = 19)
#
plotMeans(dat ~ grp,
    data = dataset, FUN_err = function(x) sd(x) * 2, pch = 15,
    ylim = c(-5, 30), yaxs = "i", connect = TRUE,
    args_con = list(lwd = 2, lty = 2, col = "grey35")
)
#
ser <- function(x) sd(x) / sqrt(length(x))
plot0(c(0, 4), c(0, 30))
plotMeans(dat ~ grp,
    data = dataset, FUN_err = ser, pch = 15,
    draw_axis = FALSE, add = TRUE, seqx = c(.5, 3.5), mar = c(6, 6, 1, 1),
    cex = 1.4
)
graphics::axis(2)
```

---

plotOnSide *Add plot on sides.*

Description

plotOnSide adds plot areas on the specified sides of the original figures.

Usage

```
plotOnSide(mat, side = 1:2, dim = NULL, quiet = FALSE, ...)
```

Arguments

mat	a matrix object specifying the location of the next N figures on the output device, see graphics::layout().
side	the number of the sides on which plot areas must be added.
dim	optional. If provided, then a matrix is created based and this argument specifies its dimensions.
quiet	if TRUE, no warning message will be displayed.
...	additional arguments to be passed to graphics::layout().

Details

This function eases the creation of plots that include multiple panels that shares information such as axis labels. Instead of repeating or deleting axis labels, `plotOnSide` add plot areas on the specified sides of the original figures. It is based on `graphics::layout()` and it is no more than a tuned version of it.

Examples

```
plotOnSide(matrix(1, 2), width = c(0.2, 1), height = c(1, 1, 1, 0.6))
graphics::layout.show(5)
```

---

pointsInPolygon *Are points inside a polygon?*

Description

For a given matrix of points coordinates, `pointsInPolygon` returns a logical vector stating whether or not these points are inside a specific polygon whose coordinate as passed as an argument.

Usage

```
pointsInPolygon(points, polygon)
```

Arguments

`points`	a matrix of coordinates of points to be tested.
`polygon`	a two-columns matrix including the coordinate of the polygon.

Details

Implements the Ray-casting algorithm.

References

https://rosettacode.org/wiki/Ray-casting_algorithm#C

Examples

```
mat <- matrix(10*runif(100), 50)
res <- pointsInPolygon(mat, cbind(c(4,8,8,4),c(4,4,8,8)))
# Visual assessment
plot0(c(0,10), c(0,10))
polygon(c(4,8,8,4),c(4,4,8,8))
graphics::points(mat[,1], mat[,2], col=res+1)
```

`polarPlot` *Polar plot*

Description

Draws a polar plot.

Usage

```
polarPlot(
  seqtime,
  seqval = NULL,
  rad = 1,
  from = 0,
  to = 2 * pi,
  incr = 0.005,
  labelc = NULL,
  tckcol = 1,
  atc = NULL,
  labelr = NULL,
  atr = NULL,
  clockwise = TRUE,
  n_signif = 2,
  add = FALSE,
  ...
)
```

Arguments

`seqtime`	sequence of time values (equivalent to the x axis).
`seqval`	sequence of values of interest (radial axis, equivalent to the y axis).
`rad`	radius of the circle.
`from`	starting point of the circle.
`to`	ending point of the circle.
`incr`	increment used to draw the circle.
`labelc`	character or expression vector of labels to be placed at the tickpoints.
`tckcol`	color of the tickmarks.
`atc`	the points at which tick-marks are to be drawn. By default (when `NULL`) tickmark locations are computed.
`labelr`	character or expression vector specifying the *text* to be placed at the radial-axis marks.
`atr`	the points at which radial-axis marks are to be drawn.
`clockwise`	logical. If TRUE, the plot must de read clockwise, otherwise, counter-clockwise.

n_signif	number of significant numbers to be displayed (used when labelc is NULL).
add	logical. add to current plot?
...	additional argument to be passed to polygon function.

Details

Polar Plot

See Also

polar.plot

Examples

```
polarPlot(1:40, stats::runif(40), to = 1.9 * pi, col = "grey30", border = "grey80")
```

---

prettyRange *Pretty range*

Description

This function returns a pretty range of values for a given a vector of type numeric.

Usage

```
prettyRange(x)
```

Arguments

x	A vector of numerical values.

Details

This function intends to generate range with round values.

Value

A vector of two values specifying the pretty the range of values.

See Also

base::pretty()

Examples

```
# Example 1:
vec <- stats::runif(20)
range(vec)
prettyRange(vec)
# Example 2:
prettyRange(c(3.849,3.88245))
```

---

ramp *Lighten or darken colors*

Description

Returns lightened or darkened colors, vectorised over `percentage`. `ramp` is valid for any couple of colors. Functions `darken()` and `lighten()` call `ramp` to respectively darken and lighten a given color.

Usage

```
ramp(fromcol, tocol, percentage = 50, as_rgb = FALSE)

darken(col, percentage = 50, as_rgb = FALSE)

lighten(col, percentage = 50, as_rgb = FALSE)
```

Arguments

`fromcol`	starting color, i.e. `percentage = 0`, it is the color returned.
`tocol`	color to nuance `fromcol`, i.e. `percentage = 100`, it is the color returned.
`percentage`	percentage determining the percentage of `tocol` used to nuance `fromcol`. Note that `darken` and `lighten` support negative percentage.
`as_rgb`	a logical. Should the color(s) returned as a matrix object?
`col`	the color to be darkened or lightened.

Functions

- `ramp()`: Returns a shaded color.
- `darken()`: Darken a color.
- `lighten()`: Lighten a color.

See Also

`grDevices::colorRampPalette()`

Examples

```
showPalette(ramp("blue", "red", 10 * 3:7))
darken("red", 50)
more_reds <- lighten("red", seq(10, 90, 9))
showPalette(more_reds)
```

`rhombi` *Rhombi*

Description

Add rhombi on a plot and returns areas.

Usage

```
rhombi(x, y = x, ldg = 1, sdg = ldg, rot = 0, add = FALSE, ...)
```

Arguments

`x`	a vector of x coordinates of the centers of the losange
`y`	a vector of y coordinates of the centers of the losange
`ldg`	vector of length of the large diagonals.
`sdg`	vector of length of the small diagonals.
`rot`	rotation angles (in degree) of the rhombi.
`add`	logical. If `TRUE` rhombi are added to the current plot (default behavior).
`...`	additional arguments to be passed to `graphics::polygon()` function.

Details

The number of rhombus maximal is provided by the length of the largest argument among x, y, ldg, sdg and rot. Other arguments are repeated with the largest length as the desired one (see `rep_len()`). Additional arguments remain the same for every rhombus.

Value

A vector including rhombi areas is returned if assigned.

Examples

```
# Example 1:
plot0(asp = 1)
rhombi(0)

# Example 2:
plot0(c(-0.4, 1.4), c(-0.4, 1.4))
rhombi(runif(6), runif(5), runif(2), runif(3))
```

```
# Example 3:
plot0(asp = 1)
rhombi(x = 0, rot = seq(0, 180, 20), col = 2, border = NA)
rhombi(x = 0, rot = seq(0, 180, 30), ldg = 0.5, col = 7, border = NA)
```

`rotation` *Rotation*

Description

Rotates a set of points.

Usage

```
rotation(x, y, rot = 90, xrot = mean(x), yrot = mean(y), rad = FALSE)
```

Arguments

x, y	x and y coordinates of points.
rot	angle of the rotation expressed in degree.
xrot, yrot	optional, x and y coordinate for the center of rotation.
rad	logical. Should radian be used rather than degrees?

Details

Returns the coordinates of the points after rotation. If the coordinates of the rotation center are not specified, then the rotation center is the centroid of the points to be rotated.

Examples

```
plot0(c(0, 10), c(0, 10))
y <- c(6, 6, 9)
x <- c(2, 5, 3.5)
polygon(x, y, lwd = 2)
myrot <- rotation(x, y, rot = 90)
polygon(myrot$x, myrot$y, lwd = 2, border = 4)
myrot2 <- rotation(x, y, rot = -40, 0, 0)
polygon(myrot2$x, myrot2$y, lwd = 2, border = 3)
```

shadowText *Shadow a Text*

Description

This function shadows a text and adds it to an existing plot.

Usage

```
shadowText(x, y, labels, col = "white", bg = par()$fg, radius = 0.1, ...)
```

Arguments

x, y	x and y coordinate of text(s) to be shadowed.
labels	A character vector to shadow.
col	color of the text.
bg	color of the background (shadow color).
radius	Width of the shadow.
...	others parameters to pass to `text()`.

Author(s)

Nicolas CASAJUS, <nicolas.casajus@gmail.com>

Examples

```
plot(1, type = "n", ann = FALSE, las = 1)
shadowText(x = 0.7, y = 1.3, labels = "This is a\nshadow text")
shadowText(x = 1.0, y = 1.3, labels = "This is a\nshadow text",
           family = "serif")
shadowText(x = 1.3, y = 1.3, labels = "This is a\nshadow text",
           family = "mono")
shadowText(x = 1.0, y = 1.0, labels = "This is a shadow text",
           family = "serif", cex = 3, col = "yellow", bg = "red")
shadowText(x = 0.7, y = 0.7, labels = "This is a\nshadow text",
           family = "serif", srt = 45)
shadowText(x = 1.0, y = 0.7, labels = "This is a\nshadow text",
           family = "serif", srt = 180)
shadowText(x = 1.3, y = 0.7, labels = "This is a\nshadow text",
           family = "serif", srt = -45)
```

showPalette *Displays a color palette*

Description

Displays a color palette and color details.

Usage

```
showPalette(
  x = palette(),
  inline = FALSE,
  add_number = TRUE,
  add_codecolor = TRUE,
  cex_num = 1.2
)
```

Arguments

x	a vector of colors.
inline	a logical. If TRUE, the colors are displayed on a single row.
add_number	a logical. If TRUE, color vector's indices are added.
add_codecolor	a logical. If TRUE, the code color is displayed.
cex_num	the magnification coefficient of the color vector's indices.

Value

The color palette displayed as an invisible output.

Examples

```
showPalette()
showPalette(inline = TRUE)
showPalette(1)
showPalette(sample(1:100, 16), add_number = FALSE, add_codecolor = FALSE)
```

stackedAreas *Stacked areas chart*

Description

Draw a stacked areas chart.

Usage

```
stackedAreas(
  val,
  index = NULL,
  rgy = 1,
  cumul = FALSE,
  transp = FALSE,
  legend = NULL,
  add = FALSE,
  col = NULL,
  pickcolors = FALSE,
  lty = 1,
  lwd = 1,
  border = NA,
  main = "",
  xlab = "",
  ylab = ""
)
```

Arguments

val	a dataframe or a matrix containing a series of positive values, rows stand for popultaions.
index	values to be used for the x axis, by default it is set to NULL meaning that it is handled by plot.default
rgy	a value that determines the range of y values. Default is set to 1 which means that the range of values is [0,1].
cumul	a logical. If TRUE, data are considered as cumulative sums.
transp	a logical. If TRUE, the transpose of the data table is computed.
legend	Text to be used as a legend for each area drawn.
add	logical. Should stacked areas be added on the current plot?
col	vector of colors, repeated if too small.
pickcolors	logical. If TRUE, pickColors() is called to select colors.
lty	the line type (see graphics::par() documentation)
lwd	the line width (see graphics::par() documentation)

border	The color to draw the border. The default, NULL, means to use graphics::par('fg'). Use border = NA to omit borders.
main	a main title for the plot.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.

Details

Areas are drawn using graphics::polygon() and users can take advantage of it to customize their stacked areas (using lwd, lty or border arguments).

Note

The default colors have been inspired by four palettes found on line: http:www.color-hex.com/color-palettes/. plotrix::stackpoly() function from the plotrix package offers a good alternative.

Using a stacked areas chart with more than 20 areas should provide a figure really hard to read.

Examples

```
# data for 8 populations at 25 different periods.
x <- data.frame(matrix(runif(200,2,10), 8, 25))

# plot 1: default plot
stackedAreas(x)

# plot 2: personalized plot
graphics::par(xaxs = 'i', yaxs = 'i', font = 2, cex.axis = 1.2,
cex.lab = 1.4, bty = 'l')
graphics::plot.default(c(1999, 2027), c(-10, 110), type = 'n',
xlab = 'Years', ylab = 'Percentage',
main = 'My customized stacked areas chart')
plotAreaColor(col = '#f2c4c4')
stackedAreas(x, index = 2001:2025, rgy = 100, lwd = 2, add = TRUE,
border = 'transparent')
```

---

textBox *Add a Box around a Text*

Description

This function adds a box around a text in an existing plot. It works differently from the plotrix::textbox() by giving more control to user (no automatic string cesure).

Usage

```
textBox(
  x,
  y,
  labels,
  align = "c",
  padding = 0,
  cex = 1,
  font = 1,
  col = par("fg"),
  family = par("family"),
  lheight = 1,
  fill = NA,
  border = par("fg"),
  density = NULL,
  angle = 45,
  lwd = par("lwd"),
  lty = par("lty")
)
```

Arguments

x	center of the box on the x-axis (see Details).
y	center of the box on the y-axis (see Details).
labels	string (of length 1) to plot and for which a box is added.
align	horizontal alignment of the text inside the box. Possible values: 'center' (or 'c'), 'left' (or 'l'), or 'right' ('r').
padding	amount of space between text limits and box border in the four directions (see Details).
cex	size of the text.
font	font of the text.
col	color of the text.
family	font family of the text
lheight	line height multiplier used to vertically space multi-line text (see Details).
fill	color to fill or shade the rectangle.
border	color of the box border.
density	density of shading lines (see `rect()`).
angle	angle (in degrees) of the shading lines (see `rect()`).
lwd	line width for box border (and box shading).
lty	line type for box border (and box shading).

Details

The xy coordinates correspond to the center of the box. If left and right paddings are identical (`padding[2] == padding[4]`) and text is centered (`align = 'c'`), then the text is also centered on these coordinates.

`padding` may be a vector of 1, 2, or 4 values, corresponding to adjustment of all box borders (1 value), top/bottom and left/right borders (2 values), or bottom/left/top/right borders (4 values). A positive value adds space between box border and text, and a negative value removes space between box border and text.

`lheight` defines the vertical inter-line spacing in a multi-line string. If `lheight = 1` (default), no inter-line spacing is added (i.e. each line string is displayed one under the other without space). If `lheight = 2`, a vertical space corresponding to one string height is added between lines. If `lheight = 0`, all line strings will be overlapping.

Only the three font families ('sans', 'serif' and 'mono') for `family` are implemented (no Hershey fonts available).

Other arguments have the same behavior as in the `rect()` (`fill` is the equivalent of `col()`) and `text()` functions.

Value

A list of length 4 with:

- **box**, the coordinates of the box (xleft, ybottom, xright, and ytop respectively);
- **labels**, the strings on each line (length of 1 if no `\n` is the original string);
- **x**, the coordinates on the x-axis of the center of each strings (length of 1 if no `\n` is the original string);
- **y**, the coordinates on the y-axis of the center of each strings (length of 1 if no `\n` is the original string).

With these coordinates, user can draw the box and the text by himself (only if same parameters are used, e.g. `cex`, `family`, `lheight`, and `font`).

Author(s)

Nicolas Casajus, <nicolas.casajus@gmail.com>

Examples

```
## Setting the scene ----
plot(1, type = "n", ann = FALSE, las = 1)
coords <- textBox(x = 1, y = 1, labels = "AqA")
str(coords)

rect(coords$box[1], coords$box[2], coords$box[3], coords$box[4], border = 3)
text(x = coords$x, y = coords$y, labels = coords$labels, col = "red")

## Padding ----
plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = "Hello World (1)",
```

```
                    padding = 0.05) # all borders
textBox(x = 1, y = 1.0, labels = "Hello World (2)",
                    padding = c(0.05, 0.20)) # bottom/top and left/right
textBox(x = 1, y = 0.8, labels = "Hello World (3)",
                    padding = c(0.05, 0.05, 0.05, 0.35)) # bottom, left, top, right

## Colors ----
plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = "Hello World (1)",
                    padding = 0.05, col = "yellow", border = "green",
                    fill = "red")

## Box Types ----
textBox(x = 1, y = 1.0, labels = "Hello World (2)",
                    padding = 0.05, lwd = 3, lty = 3)
textBox(x = 1, y = 0.8, labels = "Hello World (3)",
                    padding = 0.05, density = 30, angle = 45, fill = "gray")

## Text Fonts ----
plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = "Hello World (1)",
                    padding = 0.05, family = "mono")
textBox(x = 1, y = 1.0, labels = "Hello World (2)",
                    padding = 0.05, family = "serif")
textBox(x = 1, y = 0.8, labels = "Hello World (3)",
                    padding = 0.05, family = "serif", font = 3, cex = 3)

## Text Alignment ----
texte <- "Hello World!\nHow beautiful you are!"
plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = texte, padding = 0.05,
                    align = "l")
textBox(x = 1, y = 1.0, labels = texte, padding = 0.05,
                    align = "c")
textBox(x = 1, y = 0.8, labels = texte, padding = 0.05,
                    align = "r")

plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = texte, padding = 0.05,
                    align = "l", lheight = 0)
textBox(x = 1, y = 1.0, labels = texte, padding = 0.05,
                    align = "l", lheight = 1)
textBox(x = 1, y = 0.8, labels = texte, padding = 0.05,
                    align = "l", lheight = 2)

plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = texte, align = "l",
                    padding = c(0.05, 0.05, 0.05, 0.35))
textBox(x = 1, y = 1.0, labels = texte, align = "c",
                    padding = c(0.05, 0.05, 0.05, 0.35))
textBox(x = 1, y = 0.8, labels = texte, align = "r",
                    padding = c(0.05, 0.05, 0.05, 0.35))
```

```
## Removing Box and/or Text ----
plot(1, type = "n", ann = FALSE, las = 1)
textBox(x = 1, y = 1.2, labels = texte, col = "transparent")
textBox(x = 1, y = 1.0, labels = texte, lwd = 0)
textBox(x = 1, y = 0.8, labels = texte, lwd = 0,
                  col = "transparent")
```

`toFig` *Convert coordinates*

Description

Convert user coordinates to figure region coordinates and conversely.

Usage

```
toFig(x, y = NULL)

toUser(x, y = NULL)
```

Arguments

`x`	the x coordinates of a set of points. Alternatively, a single argument `x` can be provided.
`y`	the y coordinates of a set of points.

Details

These functions can be used anytime, however, there are useful only once `graphics::plot.new()` has been called.

Functions

- `toFig()`: Convert figure region coordinates into user coordinates.
- `toUser()`: Convert user coordinates into figure region coordinates.

translation *Translation*

Description

Compute a translation for a set of points. The transformed set of points is optionally add to the current plot.

Usage

```
translation(x, y, xtrans = 0, ytrans = 0, add = FALSE, ...)
```

Arguments

x, y	x and y coordinates of points to be translated (can also be a matrix, see details).
xtrans, ytrans	the x and y coordinates of the translation vector.
add	logical. If TRUE the set of transformed points is drawn as a polygon.
...	Additional arguments to be passed to polygon function (used only if add is TRUE).

Details

For details about what is a translation, see https://en.wikipedia.org/wiki/Translation_(geometry). Note that if x is a matrix with more than 2 columns, then x is the first column and y the second one.

Examples

```
# Example 1:
plot0(c(0,10),c(0,10))
x <- c(4,6,5)
y <- c(2,2,4)
polygon(x,y)
trans1 <- translation(x,y,xtrans=2,ytrans=5, add=TRUE, border=4, lwd=2)

# Example 2:
x <- c(2,4,3,1)
y <- c(1,1,3,3)
plot0(c(0,10),c(0,10))
polygon(x,y)
for (i in seq_len(6)) translation(x,y,xtrans=i,ytrans=i, add=TRUE,
border=i+1, lwd=2)
```

vecfield2d *Vector Fields.*

Description

Draw a vector field associated to a system of at least two ODE.

Usage

```
vecfield2d(
  coords,
  FUN,
  args = NULL,
  ndim = NULL,
  slices = c(1, 2),
  fixed = NULL,
  cex.x = 0.25,
  cex.y = cex.x,
  log = FALSE,
  add = FALSE,
  ...
)
```

Arguments

coords	a matrix with two columns or more that is optionally used to alternatively define the coordinates of the vector field.
FUN	the function that describes the dynamical system (see details).
args	the parameters of the dynamical system (see details).
ndim	number of dimension of the system. If NULL the values is based on coords and slice
slices	a vector of 2 elements providing the dimensions to be displayed, (default set to c(1,2)).
fixed	the values used for non drawn dimension, if NULL the values will be set to 0.
cex.x	the magnification coefficient to be used for lengths of vectors along the x axis.
cex.y	the magnification coefficient to be used for lengths of vectors along the y axis.
log	a logical. If TRUE, the lengths of arrows are log-transformed.
add	a logical. If TRUE, the vector field is added on the current plot.
...	additional arguments to be passed to arrows().

Details

The FUN function must be a function of at least two arguments. The first argument must contain the dynamical variables as a vector and the second arguments must contain all the other parameters that shapes the dynamical system. When some dimensions are missing, the order of fixed is the one in FUN once the drawn dimension are withdrawn.

Examples

```
systLin <- function(X, beta){
    Y <- matrix(0,ncol=2)
    Y[1L] <- beta[1,1]*X[1L]+beta[1,2]*X[2L]
    Y[2L] <- beta[2,1]*X[1L]+beta[2,2]*X[2L]
    return(Y)
}
seqx <- seq(-2,2,0.31)
seqy <- seq(-2,2,0.31)
beta1 <- matrix(c(0,-1,1,0),2)
# Plot 1:
vecfield2d(coords=expand.grid(seqx, seqy), FUN=systLin,
args=list(beta=beta1))
# Plot 2:
graphics::par(mar=c(2,2,2,2))
vecfield2d(coords=expand.grid(seqx, seqy), FUN=systLin,
   args=list(beta=beta1), cex.x=0.35, cex.arr=0.25,
   border=NA,cex.hh=1, cex.shr=0.6, col=8)
graphics::abline(v=0,h=0)
```

Index

∗ **Gantt**
dfGantt, 17
∗ **angles,**
getAngle2d, 24
∗ **areas,**
stackedAreas, 45
∗ **arrows**
arrows2, 6
∗ **axis,**
addAxis, 3
∗ **background,**
plotAreaColor, 33
∗ **background**
addGrid, 5
∗ **box,**
addFrame, 4
addGrid, 5
∗ **boxplots**
biBoxplot, 8
∗ **box**
box2, 9
frameIt, 22
textBox, 46
∗ **character**
pchImage, 30
∗ **circle**
circles, 12
∗ **color,**
pickColors, 32
showPalette, 44
∗ **colors,**
contrastColors, 16
gpuPalette, 25
∗ **color**
plotAreaColor, 33
∗ **contrast**
contrastColors, 16
∗ **datasets**
dfGantt, 17
gpuPalettes, 25
∗ **deviation**
plotMeans, 35
∗ **donut**
donut, 17
∗ **ellipse**
ellipse, 19
∗ **empty**
plot0, 32
vecfield2d, 52
∗ **frame,**
addFrame, 4
frameIt, 22
∗ **geometries**
circles, 12
∗ **geometry**
homothety, 26
translation, 51
∗ **grid,**
addGrid, 5
∗ **grid**
addFrame, 4
∗ **histogram**
stackedAreas, 45
∗ **homothety,**
homothety, 26
∗ **image,**
pchImage, 30
∗ **image**
image2, 27
∗ **interactive**
interactiveLayout, 29
pickColors, 32
∗ **interval.**
prettyRange, 39
∗ **layout**
interactiveLayout, 29
∗ **means,**
plotMeans, 35

∗ **over**
pointsInPolygon, 37
∗ **palette,**
pickColors, 32
∗ **palette**
gpuPalette, 25
∗ **plot,**
pchImage, 30
stackedAreas, 45
∗ **plotting**
pchImage, 30
∗ **plot**
plot0, 32
vecfield2d, 52
∗ **points,**
pointsInPolygon, 37
∗ **polygons,**
pointsInPolygon, 37
∗ **range,**
prettyRange, 39
∗ **rectangles**
image2, 27
∗ **rhumbus**
rhombi, 41
∗ **rotation**
rotation, 42
∗ **selection**
showPalette, 44
∗ **shadow**
shadowText, 43
∗ **spatial**
addAxis, 3
∗ **standard**
plotMeans, 35
∗ **text,**
shadowText, 43
textBox, 46
∗ **translation,**
translation, 51
∗ **vectors**
getAngle2d, 24

addAxis, 3
addFrame, 4
addGrid, 5
arrows(), *52*
arrows2, 6
as.Date(), *23*
base::pretty(), *39*
base::rep(), *15*
biBoxplot, 8
blendColors, 9
box2, 9
boxplot2, 10

circles, 12
col2Hex *(*contrastColors*)*, 16
colorScale, 13
compassRose, 14
compassRoseCardinal *(*compassRose*)*, 14
contrastColors, 16

darken *(*ramp*)*, 40
dfGantt, 17
donut, 17

ellipse, 19
encircle, 20
envelop, 21

frameIt, 22

ganttChart, 23
ganttChart(), *17*
getAngle2d, 24
gpuPalette, 25
gpuPalettes, 25
graphics::arrows(), *7*
graphics::axis(), *28*
graphics::box(), *10*
graphics::boxplot(), *8*
graphics::image(), *27, 28*
graphics::layout(), *29*, *36, 37*
graphics::layout.show(), *29*
graphics::lines(), *8*
graphics::par(), *9*, *33, 34*, *45*
graphics::pie(), *18*
graphics::plot.default, *33*
graphics::plot.new(), *50*
graphics::polygon(), *12*, *15*, *19–21*, *26*, *41*, *46*
graphics::rect(), *28*, *34*
graphics::symbols(), *13*
graphics::title(), *28*
graphicsutils, 26
grDevices::chull(), *20*
grDevices::col2rgb(), *16*

grDevices::colorRampPalette(), *25*, *40*

homothety, 26
howManyRC, 27

image2, 27
interactiveLayout, 29

jpeg::readJPEG(), *30*

lighten *(*ramp*)*, 40
lighten(), *23*

pchImage, 30
percX, 31
percY *(*percX*)*, 31
pickColors, 32
pickColors(), *45*
plot.default(), *36*
plot0, 32
plotAreaColor, 33
plotImage, 34
plotMeans, 35
plotOnSide, 36
plotrix::draw.arc(), *13*
plotrix::draw.circle(), *13*
plotrix::stackpoly(), *46*
png::readPNG(), *30*
pointsInPolygon, 37
polar.plot, *39*
polarPlot, 38
prettyRange, 39

ramp, 40
rep_len(), *12*, *19*, *41*
rhombi, 41
rotation, 42

shadowText, 43
showPalette, 44
stackedAreas, 45
stats::aggregate(), *11*
stats::formula(), *35*
stats::quantile(), *8*
stats::sd(), *35*

textBox, 46
toFig, 50
toUser *(*toFig*)*, 50
translation, 51

vecfield2d, 52