# Package 'pharos'

November 25, 2026

**Title** Descriptive Statistics Graphics and Utilities

**Version** 0.1.6

**Description** Draws the statistical graphics of the 'DescToolsX' ecosystem on top of base graphics: distribution and density displays, bivariate and categorical plots, diagnostic and model evaluation panels, and a faceting mechanism for arranging them. A theme system resolves colours, symbols and layout centrally, so a consistent look can be set once rather than repeated at every call. The package also contains the colour conversion and palette tools, the geometry helpers, the number and string formatting routines and the HTML output utilities that the plots are built from. It is self contained and usable without the rest of the suite.

**Depends** R (>= 4.6.0)

**License** GPL (>= 2)

**Encoding** UTF-8

**LinkingTo** Rcpp

**Imports** Rcpp, bedrock (>= 0.1.18), MASS, cli, withr, stringi, methods, base64enc, grid

**Suggests** testthat (>= 3.0.0), R.rsp, hexbin, prettyunits, pdftools

**Config/testthat/edition** 3

**VignetteBuilder** R.rsp

**URL** https://andrisignorell.github.io/pharos/, https://github.com/AndriSignorell/pharos/

**BugReports** https://github.com/AndriSignorell/pharos/issues

**Config/roxygen2/version** 8.1.1

**NeedsCompilation** yes

**Author** Andri Signorell [aut, cre] (ORCID: <https://orcid.org/0000-0003-4311-1969>), R Core Team [ctb], Ben Bolker [ctb],

Jeff Enos [ctb],
Maxime Genest [ctb] (Fortran code of package 'depth'),
Frank E. Harrell Jr [ctb],
Henk Harmsen [ctb],
Christian W. Hoffmann [ctb],
Markus Huerzeler [ctb],
Matthias Kohl [ctb],
Jim Lemon [ctb],
Arni Magnusson [ctb],
Jean-Claude Massé [ctb] (Fortran code of package 'depth'),
Markus Naepflin [ctb],
Jean-François Plante [ctb] (Fortran code of package 'depth'),
B. D. Ripley [ctb],
Peter J. Rousseeuw [ctb, cph] (Algorithm AS 307 (halfspace depth)),
Ida Ruts [ctb, cph] (Algorithm AS 307 (halfspace depth)),
Terry Therneau [ctb],
John Verzani [ctb],
W. N. Venables [ctb],
Hans Peter Wolf [ctb],
Ying Wu [ctb]

**Maintainer** Andri Signorell <andri@signorell.net>

**Repository** CRAN

**Date/Publication** 2026-11-25 12:20:02 UTC

# Contents

abcCoords . . . . . 5
addOpacity . . . . . 7
arc . . . . . 8
as.CI . . . . . 9
asFileLink . . . . . 12
asHtml . . . . . 13
asImg . . . . . 14
axisBreak . . . . . 15
axisFmt . . . . . 16
axTicksPosixct . . . . . 18
band . . . . . 19
barText . . . . . 20
bezier . . . . . 23
boxedText . . . . . 24
canvas . . . . . 26
circle . . . . . 27
cmykToCmy . . . . . 28
cmykToRgb . . . . . 28
cmyToCmyk . . . . . 29
colLegend . . . . . 30
color-conversion-overview . . . . . 32

colToHex . . . 34
colToHsv . . . 34
colToOpaque . . . 35
colToRgb . . . 36
contrastCol . . . 37
convUnit . . . 38
coordinate-conversions . . . 40
darken . . . 41
degree-radians-conversion . . . 42
ellipse . . . 43
embedFile . . . 44
errBars . . . 44
escapeHtml . . . 47
exportPlot . . . 48
fade . . . 50
findCol . . . 51
fm . . . 53
fmCI . . . 59
ftable.list . . . 60
grayScale . . . 61
hcol . . . 62
hexToCol . . . 63
hexToRgb . . . 64
htmlNotation . . . 64
htmlSubscript . . . 65
isValidPlotRegion . . . 66
lighten . . . 67
lines.lm . . . 68
lines.loess . . . 70
lineSep . . . 71
lineToUser . . . 72
longToRgb . . . 73
mar . . . 74
mixCol . . . 75
pal . . . 76
palNames . . . 77
plot.BlandAltman . . . 78
plot.Desc.qn . . . 80
plot.Desc.table . . . 82
plot.Lc . . . 85
plotArea . . . 88
plotAssoc . . . 90
plotBag . . . 93
plotBar . . . 97
plotBinaryTree . . . 100
plotBox . . . 102
plotBubble . . . 105
plotCatDist . . . 107

plotCirc . . . 109
plotCor . . . 111
plotDens . . . 115
plotDens2D . . . 117
plotDensBox . . . 119
plotDot . . . 122
plotECDF . . . 125
plotFacet . . . 128
plotFreqDist . . . 131
plotFun . . . 134
plotHeatmap . . . 136
plotHexbin . . . 138
plotLift . . . 140
plotLines . . . 143
plotMiss . . . 145
plotMosaic . . . 146
plotPolar . . . 148
plotProbDist . . . 150
plotPropCI . . . 152
plotQQ . . . 154
plotRidge . . . 156
plotSankey . . . 159
plotTernary . . . 162
plotTimeSeries . . . 163
plotTreemap . . . 164
plotViolin . . . 166
plotWeb . . . 170
plotXY . . . 172
polarGrid . . . 175
polygonX . . . 177
Prefix . . . 178
preview . . . 179
preview.html . . . 180
print.Unit . . . 181
regPolygon . . . 182
rgbToCmy . . . 183
rgbToCol . . . 183
rgbToHex . . . 184
rgbToLong . . . 185
ring . . . 186
rotate . . . 187
setBackCol . . . 188
shade . . . 189
splineCI . . . 190
spreadOut . . . 192
stamp . . . 193
strAbbr . . . 194
strAlign . . . 195

strCap . . . . . 196
strChop . . . . . 197
strCountWords . . . . . 199
strDist . . . . . 199
strExtract . . . . . 201
strExtractBetween . . . . . 202
string-overview . . . . . 203
strIsNumeric . . . . . 204
strLeftRight . . . . . 205
strLen . . . . . 206
strPad . . . . . 207
strPos . . . . . 208
strRev . . . . . 209
strSpell . . . . . 210
strSplit . . . . . 211
strTrim . . . . . 212
strTrunc . . . . . 213
strVal . . . . . 214
style . . . . . 215
textLegend . . . . . 219
theme . . . . . 220
titleRect . . . . . 224
toHtmlTable . . . . . 225
transformXY . . . . . 226
unit . . . . . 228

**Index** **230**

---

`abcCoords` *Coordinates for Named Plot Positions*

---

## Description

Returns the xy-coordinates and text-adjustment values for named anchor positions such as `"topleft"`, `"center"`, etc., as used by `legend()`. Useful for placing text or other annotations at consistent, region-aware positions.

## Usage

```
abcCoords(
  x = "topleft",
  region = c("plot", "figure", "device"),
  cex = NULL,
  inset = 0
)
```

## Arguments

| | |
|---|---|
| x | A character string specifying the anchor position. One of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", or "center". Partial matching is supported. |
| region | one of "plot" (default), "figure", or "device". Determines the coordinate region used. See Details for the clipping that comes with the latter two. |
| cex | character expansion factor. If NULL (default), the current par("cex") is used. |
| inset | inset distance from the boundary, specified in lines of text (character heights/widths). May be a scalar (applied to both x and y) or a length-2 vector (x inset, y inset). Default 0. |

## Details

The positioning logic is adapted from legend(). The inset is computed in character units via strwidth() and strheight(), making it robust to device resizing, font changes, and plot-range scaling.

Three regions are supported:

"plot" The inner plot area (par("usr")). The default, and the region legend() positions in.

"figure" The figure region within the device, accounting for par("fig").

"device" The full device area.

Coordinates for "figure" and "device" lie outside par("usr") and are therefore clipped away by text() and friends unless the drawing call sets xpd = NA. The returned values are in user space either way; only the clipping has to be turned off by the caller.

## Value

A list with two components:

xy A list with elements x and y giving the anchor coordinates in user space.

adj A numeric vector of length 2, suitable for the adj argument of text().

## See Also

text(), legend()

Other graphics.layout: axTicksPosixct(), axisBreak(), isValidPlotRegion(), lineToUser(), mar(), plotFacet(), spreadOut()

## Examples

```
plot(x = rnorm(10), type = "n", xlab = "", ylab = "")

# coordinates only
abcCoords("bottomleft")

# place text at a named position
xy <- abcCoords("bottomleft")
```

```
text(x = xy$xy$x, y = xy$xy$y, labels = "My Label", adj = xy$adj)

# all nine positions with inset
sapply(c("topleft", "top", "topright",
         "left",    "center", "right",
         "bottomleft", "bottom", "bottomright"),
  function(p) {
    xy <- abcCoords(p, inset = 1)
    text(x = xy$xy$x, y = xy$xy$y,
         labels = p, adj = xy$adj)
  })

# the margins need xpd = NA, whatever the coordinates say
xy <- abcCoords("bottomright", region = "figure", inset = 0.5)
text(x = xy$xy$x, y = xy$xy$y, labels = "source: ...",
     adj = xy$adj, cex = 0.7, xpd = NA)
```

---

`addOpacity` *Add an Alpha Channel to Colors*

---

## Description

Add transparency to colors.

## Usage

```
addOpacity(col, opacity = 0.5)
```

## Arguments

| | |
|---|---|
| `col` | vector of valid R colors |
| `opacity` | opacity value between 0 and 1 |

## Value

Character vector of hexadecimal colors with alpha channel.

## See Also

grDevices::adjustcolor

Other color.manipulation: `colToOpaque()`, `darken()`, `fade()`, `lighten()`, `mixCol()`

## Examples

```
op <- par(no.readonly = TRUE)

addOpacity("yellow", 0.2)
addOpacity(2, 0.5)   # red

canvas(3)
polygonX(circle(x=c(-1,0,1), y=c(1,-1,1), radius=2),
        col=addOpacity(2:4, 0.4))

x <- rnorm(15000)
par(mfrow=c(1,2))
plot(x, type="p", col="blue" )
plot(x, type="p", col=addOpacity("blue", .2),
     main="Better insight with alpha channel" )

par(op)
```

---

`arc` *Arc Geometry*

---

## Description

Create one or more circular or elliptic arcs.

## Usage

```
arc(
  x = 0,
  y = 0,
  radiusX = 1,
  radiusY = radiusX,
  startAngle = 0,
  endAngle = 2 * pi,
  nPoints = 100
)
```

## Arguments

| | |
|---|---|
| `x`, `y` | coordinates of the arc centre |
| `radiusX`, `radiusY` | horizontal and vertical radius |
| `startAngle`, `endAngle` | start and end angle in radians |
| `nPoints` | number of points used to approximate the arc |

## Value

An object inheriting from class "arcGeometry".

## See Also

Other geometry.structures: band(), bezier(), circle(), ellipse(), polygonX(), regPolygon(), ring()

## Examples

```
canvas(xlim = c(-2, 2))

# a quarter of a circle
lines(arc(radiusX = 1.5, startAngle = 0, endAngle = pi/2), lwd = 2)

# elliptic arcs, vectors of radii yield a collection of arcs
a <- arc(radiusX = c(0.5, 1), radiusY = c(1, 1.8),
         startAngle = pi, endAngle = 3*pi/2)
lines(a, col = "red")
```

---

# as.CI *Confidence Interval Objects*

---

## Description

Converts common confidence interval representations into a standardized object of class "CI". The standardized representation removes the ambiguity between ordinary numeric data and confidence interval data.

## Usage

```
as.CI(x, ...)

## S3 method for class 'matrix'
as.CI(x, ...)

## S3 method for class 'data.frame'
as.CI(x, estimate = "est", lower = "lci", upper = "uci", ...)

## S3 method for class 'list'
as.CI(x, ...)

## S3 method for class 'CI'
as.CI(x, ...)

## Default S3 method:
as.CI(x, ...)
```

```
isCI(x)
```

## Arguments

| | |
|---|---|
| x | object to convert or, for isCI(), object to test |
| ... | further arguments passed to methods |
| estimate | name of the data-frame column containing the point estimates |
| lower | name of the data-frame column containing the lower confidence limits |
| upper | name of the data-frame column containing the upper confidence limits |

## Details

A "CI" object is a data frame containing the columns est, lci, and uci. Additional columns are retained and can be used as grouping variables by functions such as plotDot().

The primary purpose of as.CI() is to declare explicitly that an object contains estimates and confidence limits. For example, a numeric matrix with three columns is normally ambiguous: its columns may represent three groups or the estimate, lower limit, and upper limit. Passing the matrix to as.CI() declares that its columns have the latter meaning.

Supported inputs are:

- a numeric matrix with exactly three columns, interpreted in the order est, lci, and uci
- a data frame containing columns for the estimates and confidence limits; their names can be specified with estimate, lower, and upper
- a list in which every element contains three values representing c(est, lci, uci)
- an array-like result from tapply() in which every cell contains c(est, lci, uci); its dimensions are converted to grouping variables
- an existing "CI" object, which is returned unchanged

The standardized object can be passed directly to plotDot() to display the estimates and their confidence intervals. This is particularly useful for matrices, because a bare matrix supplied to plotDot() is interpreted as grouped estimates rather than as confidence interval data.

## Value

as.CI() returns a data frame of class "CI" containing the columns est, lci, and uci, followed by any grouping columns; isCI() returns a single logical value

## See Also

plotDot(), fmCI()

## Examples

```
# matrix containing estimate, lower limit, and upper limit
x <- matrix(
  c(
    10, 20, 30,
     8, 18, 28,
    12, 22, 32
  ),
  ncol = 3,
  dimnames = list(
    c("A", "B", "C"),
    c("est", "lci", "uci")
  )
)

ci <- as.CI(x)
ci
isCI(ci)

# display the estimates and confidence intervals
plotDot(ci)

# data frame using the standard column names
d <- data.frame(
  est = c(10, 20),
  lci = c(8, 18),
  uci = c(12, 22),
  sex = c("F", "M")
)

as.CI(d)

# data frame using different column names
d <- data.frame(
  item = c("A", "B"),
  estimate = c(10, 20),
  lower = c(8, 18),
  upper = c(12, 22)
)

as.CI(
  d,
  estimate = "estimate",
  lower = "lower",
  upper = "upper"
)

# confidence intervals returned by tapply(): every cell holds the
# estimate and its limits (as e.g. lumen::meanCI() returns them)
meanCI <- function(z) {
  ci <- t.test(z)$conf.int
  c(est = mean(z), lci = ci[1], uci = ci[2])
```

```
}

xci <- with(
  warpbreaks,
  tapply(
    breaks,
    list(wool, tension),
    meanCI
  )
)

as.CI(xci)
plotDot(as.CI(xci))
```

---

`asFileLink` *Link to a self-contained embedded file*

---

## Description

Turns a file into a download link that carries the file with it: the contents travel base64-encoded inside the `href`, so the resulting HTML needs no server and no accompanying assets. The counterpart of `asImg()` for non-image files – a data set next to a table, a script next to its output.

## Usage

```
asFileLink(path, label = NULL, type = NULL)
```

## Arguments

| | |
|---|---|
| `path` | path to an existing file |
| `label` | link text; defaults to the file name |
| `type` | MIME type; guessed from the extension when `NULL` |

## Details

Browsers limit the size of a data URI, so this suits spreadsheets and text files rather than large binaries.

## Value

an object of class `c("html", "character")`

## See Also

Other html: `asHtml()`, `asImg()`, `embedFile()`, `escapeHtml()`, `htmlNotation`, `htmlSubscript`, `preview()`, `preview.html()`, `toHtmlTable()`

## Examples

```
fn <- tempfile(fileext = ".csv")
write.csv(head(iris), fn, row.names = FALSE)
asFileLink(fn, label = "iris")
unlink(fn)
```

---

`asHtml` *Mark a character vector as HTML*

---

## Description

Tags a character vector with the S3 class `"html"` so that it prints via `preview.html()` as readable text instead of as a raw character vector.

## Usage

```
asHtml(x)
```

## Arguments

| | |
|---|---|
| `x` | a character vector, typically containing HTML markup |

## Value

`x`, with class `"html"` added

## See Also

Other html: `asFileLink()`, `asImg()`, `embedFile()`, `escapeHtml()`, `htmlNotation`, `htmlSubscript`, `preview()`, `preview.html()`, `toHtmlTable()`

## Examples

```
asHtml("<b>bold</b>")
```

asImg *Embed a plot as an inline HTML image*

## Description

Evaluates a plotting expression in a temporary PNG device and returns the resulting image as a self-contained, base64-encoded <img> tag (class "html", see asHtml()), suitable for embedding directly in HTML text – a report, a question, an e-mail.

## Usage

```
asImg(expr, width = 520, height = 440, res = 96, ...)
```

## Arguments

| | |
|---|---|
| expr | a plotting expression, evaluated once inside the device. A character string is not accepted: pass the plotting code itself, e.g. asImg(plot(1:10)) instead of asImg("plot(1:10)"). |
| width, height | size of the device in pixels |
| res | nominal resolution in dpi, which also scales the text: raise it for a larger picture with the same relative proportions |
| ... | further arguments passed to grDevices::png() |

## Details

The expression is passed unevaluated and carries its own environment, so a plot built inside a function sees that function's local variables. Several statements are given in braces, as in the examples below.

## Value

an object of class c("html", "character") containing an <img> tag with a ... source

## See Also

Other html: asFileLink(), asHtml(), embedFile(), escapeHtml(), htmlNotation, htmlSubscript, preview(), preview.html(), toHtmlTable()

## Examples

```
img <- asImg(plot(1:10))

# several statements, and local variables
f <- function(n) {
  x <- seq_len(n)
  asImg({
```

```
    plot(x, x^2, type = "b")
    abline(h = mean(x^2), lty = 2)
  })
}
substr(f(10), 1, 30)
```

---

axisBreak *Place a Break Mark on an Axis*

---

## Description

Places a break mark on an axis on an existing plot.

## Usage

```
axisBreak(
  axis = 1,
  breakpos = NULL,
  pos = NA,
  bgcol = "white",
  breakcol = "black",
  style = "slash",
  brw = 0.02
)
```

## Arguments

| | |
|---|---|
| axis | which axis to break |
| breakpos | where to place the break in user units |
| pos | position of the axis (see axis) |
| bgcol | the color of the plot background |
| breakcol | the color of the "break" marker |
| style | either 'gap', 'slash' or 'zigzag' |
| brw | break width relative to plot width |

## Details

The 'pos' argument is not needed unless the user has specified a different position from the default for the axis to be broken.

## Value

No return value, called for its side effect of drawing the break mark on the current plot.

## Note

There is some controversy about the propriety of using discontinuous coordinates for plotting, and thus axis breaks. Discontinuous coordinates allow widely separated groups of values or outliers to appear without devoting too much of the plot to empty space.
The major objection seems to be that the reader will be misled by assuming continuous coordinates. The 'gap' style that clearly separates the two sections of the plot is probably best for avoiding this.

Based on code by Jim Lemon and Ben Bolker, adapted to conform to package standards

## See Also

Other graphics.layout: `abcCoords()`, `axTicksPosixct()`, `isValidPlotRegion()`, `lineToUser()`, `mar()`, `plotFacet()`, `spreadOut()`

## Examples

```
plot(3:10, main="Axis break test")

# put a break at the default axis and position
axisBreak()
axisBreak(2, 2.9, style="zigzag")
```

---

`axisFmt` *Draw an Axis With Formatted or Rotated Labels*

---

## Description

A drop-in replacement for `graphics::axis()` that adds two conveniences: tick labels can be formatted through `fm()` via the `fmt` argument, and labels can be drawn at an arbitrary angle using `srt` (which `graphics::axis()` itself ignores). When rotated labels are requested the function also estimates the margin space they require and, optionally, widens the corresponding plot margin so the labels are not clipped.

## Usage

```
axisFmt(
  side,
  at = NULL,
  fmt = NULL,
  labels = TRUE,
  srt = NULL,
  adj = NULL,
  estimateMar = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| side | integer specifying which side of the plot the axis is drawn on, as in graphics::axis(): 1 = below, 2 = left, 3 = above and 4 = right. |
| at | numeric vector of tick positions. If NULL (default) the positions are determined with graphics::axTicks(). |
| fmt | format specification passed to fm() to format the tick labels. If NULL (default) the labels are used as they are (or the bare tick values, coerced to character). See Details. |
| labels | either a logical or a character vector of labels. If TRUE (default) the labels are generated from at (formatted with fmt if supplied). A character vector is used verbatim. |
| srt | numeric, the string rotation angle in degrees. If NULL (default) labels are drawn horizontally through the ordinary graphics::axis() mechanism. Any other value triggers the rotated-label path described in Details (a common choice for long category names is srt = 45). |
| adj | label justification passed to graphics::text(). If NULL a sensible default is chosen from side and the rotation: right aligned with the tick for the x-axes (c(1, 0.5)) and bottom aligned for the y-axes (c(0.5, 0)). |
| estimateMar | logical. If TRUE (default) and srt is set, the margin on side is temporarily widened to the estimated space needed by the rotated labels (restored on exit). Set to FALSE to leave the margins untouched. |
| ... | further arguments passed to both graphics::axis() (for the line and ticks) and graphics::text() (for the labels). Graphical parameters shared by the two (e.g. col, cex) therefore affect both. |

## Details

The fmt argument is passed straight to fm() and therefore accepts the full range of format specifications: a special short code (e.g. "\%", "e", "eng", "p"), an ISO-8601 date pattern (e.g. "MMM yyyy"), a Style object, a bare (named) list treated as a style template (e.g. fmt = list(digits = 1, bigMark = " ")), or a function of x. See fm() for the details.

When srt is set, the axis line and ticks are drawn without labels via graphics::axis(), and the labels are added separately with graphics::text() using xpd = TRUE so they may extend into the figure margin. The required margin width is estimated by projecting the rotated label boxes onto the axis normal (see estimateMar).

Note that changing par("mar") after the plot region has been established does not resize the existing region. When estimateMar is TRUE the widened margin therefore mainly ensures enough device space outside the plot region for the (xpd = TRUE) labels. For a clean layout, set the margin *before* calling graphics::plot(); the returned mar component reports the estimated requirement so a calling routine can reserve the space in advance.

## Value

invisibly, a list with components at (the tick positions used) and mar (the estimated margin requirement in lines for the rotated labels, or NA when srt is NULL).

## Author(s)

Andri Signorell andri@signorell.net

## See Also

graphics::axis, graphics::axTicks, graphics::text, fm

## Examples

```
# formatted tick labels
plot(1:10, runif(10) * 1000, yaxt = "n", xaxt = "n")
axisFmt(2, fmt = list(digits = 0, bigMark = " "))
axisFmt(1, fmt = list(fmt = "%", digits = 1))

# rotated category labels (margin widened automatically)
plot(1:5, c(7, 6, 11, 5, 12), xaxt = "n", xlab = "")
axisFmt(1, at = 1:5, labels = paste("Category", LETTERS[1:5]), srt = 45)
```

---

# axTicksPosixct *Compute Axis Tickmark Locations (For POSIXct Axis)*

---

## Description

Compute pretty tickmark locations, the same way as R does internally. By default, gives the at values which axis.POSIXct(side, x) would use.

## Usage

```
axTicksPosixct(side, x, at, format, labels = TRUE, ...)

axTicksDate(side = 1, x, ...)
```

## Arguments

| | |
|---|---|
| side | see graphics::axis |
| x, at | date-time or date object. |
| format | see base::strptime |
| labels | either a logical value specifying whether annotations are to be made at the tick-marks, or a vector of character strings to be placed at the tickpoints |
| ... | further arguments to be passed from or to other methods |

## Details

graphics::axTicks() has no implementation for POSIXct axis. This function fills the gap.

## Value

numeric vector of coordinate values at which axis tickmarks can be drawn.

## See Also

graphics::axTicks(), graphics::axis.POSIXct()

Other graphics.layout: abcCoords(), axisBreak(), isValidPlotRegion(), lineToUser(), mar(), plotFacet(), spreadOut()

## Examples

```
with(beaver1, {
  time <- strptime(paste(1990, day, time %/% 100, time %% 100),
                   "%Y %j %H %M")
  plot(time, temp, type = "l") # axis at 4-hour intervals.
  # now label every hour on the time axis
  plot(time, temp, type = "l", xaxt = "n")
  r <- as.POSIXct(round(range(time), "hours"))
  axis.POSIXct(1, at = seq(r[1], r[2], by = "hour"), format = "%H")
  # place the grid
  abline(v=axTicksPosixct(1, at = seq(r[1], r[2], by = "hour"), format = "%H"),
         col="grey", lty="dotted")
})
```

---

band *Band Geometry*

---

## Description

Create a polygonal band from upper and lower boundaries.

## Usage

```
band(x, y)
```

## Arguments

| | |
|---|---|
| x | A vector or matrix of x coordinates. |
| y | A vector or matrix of y coordinates. |
| | If either x or y is supplied as a two-column matrix, the second column is interpreted as the lower boundary and reversed automatically. |

## Details

Typically used to represent confidence or prediction bands.

## Value

An object inheriting from class "bandGeometry".

## See Also

graphics::polygon

Other geometry.structures: arc(), bezier(), circle(), ellipse(), polygonX(), regPolygon(), ring()

## Examples

```
set.seed(18)

x <- rnorm(15)
y <- x + rnorm(15)

new <- seq(-3, 3, 0.5)

pred <- predict(
  lm(y ~ x),
  newdata = data.frame(x = new),
  interval = "confidence"
)

plot(y ~ x)

polygonX(
  band(
    x = new,
    y = pred[,2:3]
  ),
  col = addOpacity("grey80", 0.5),
  border = NA
)
```

---

# barText *Place Value Labels on a Barplot*

---

## Description

It can sometimes make sense to display data values directly on the bars in a barplot. There are a handful of obvious alternatives for placing the labels, either on top of the bars, right below the upper end, in the middle or at the bottom. Determining the required geometry - although not difficult - is cumbersome and the code is distractingly long within an analysis code. The present function offers a short way to solve the task. It can place text either in the middle of the stacked bars, on top or on the bottom of a barplot (side by side or stacked).

## Usage

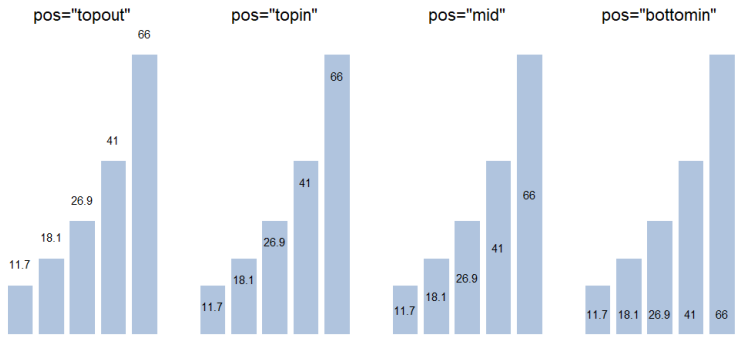
```
barText(
  height,
  mids,
  labels = height,
  beside = FALSE,
  horiz = FALSE,
  cex = par("cex"),
  adj = NULL,
  pos = c("top-out", "top-in", "mid", "bottom-in", "bottom-out"),
  offset = 0,
  col = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| height | either a vector or matrix of values describing the bars which make up the plot exactly as used for creating the barplot. |
| mids | the returned mid points as returned by b <- barplot(...). |
| labels | the labels to be placed on the bars. |
| beside | a logical value. If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars. |
| horiz | a logical value. If FALSE, the bars are drawn vertically with the first bar to the left. If TRUE, the bars are drawn horizontally with the first at the bottom. |
| cex | numeric character expansion factor; multiplied by par()("cex") yields the final character size. NULL and NA are equivalent to 1.0. |
| adj | one or two values in [0, 1] which specify the x (and optionally y) adjustment of the labels. On most devices values outside that interval will also work. |
| pos | one of "top-out", "top-in", "mid", "bottom-in", "bottom-out", defining if the labels should be placed on top of the bars (inside or outside) or at the bottom of the bars (inside or outside). |
| offset | a vector indicating how much the bars should be shifted relative to the x axis. |
| col | the color and to be used, possibly a vector (default in par("col")). |
| ... | the dots are passed to the boxedText(). |

## Details

The x coordinates of the labels can be found by using barplot() result, if they are to be centered at the top of each bar. barText() calculates the rest.




Notice that when the labels are placed on top of the bars, they may be clipped. This can be avoided by setting xpd=TRUE.

## Value

returns the geometry of the labels invisibly

## See Also

Other graphics.annotation: boxedText(), colLegend(), errBars(), stamp(), textLegend(), titleRect()

## Examples

```
# simple vector
x <- c(353, 44, 56, 34)
b <- barplot(x)
barText(x, b, x)

# more complicated
b <- barplot(VADeaths, horiz = FALSE, col="steelblue", beside = TRUE)
barText(VADeaths, mids=b, horiz = FALSE, beside = TRUE, cex=0.8)
barText(VADeaths, mids=b, horiz = FALSE, beside = TRUE, cex=0.8, pos="bottom-in",
        col="white", font=2)

b <- barplot(VADeaths, horiz = TRUE, col="steelblue", beside = TRUE)
barText(VADeaths, mids=b, horiz = TRUE, beside = TRUE, cex=0.8)

b <- barplot(VADeaths)
barText(VADeaths, mids=b)

b <- barplot(VADeaths, horiz = TRUE)
barText(VADeaths, mids=b, horiz = TRUE, col="red", cex=1.5)
```

`bezier` *Bézier Geometry*

## Description

Create a Bézier curve from a set of control points.

## Usage

```
bezier(x, y = NULL, nPoints = 100)
```

## Arguments

| | |
|---|---|
| `x, y` | numeric vectors of control points. Alternatively, `x` may be a list with components `x` and `y`. |
| `nPoints` | number of points used to approximate the curve. |

## Value

An object inheriting from class `"bezierGeometry"`.

## References

Farin, G. (1993). *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press.

## See Also

graphics::lines

Other geometry.structures: `arc()`, `band()`, `circle()`, `ellipse()`, `polygonX()`, `regPolygon()`, `ring()`

## Examples

```
canvas(xlim = c(0, 1))

lines(
  bezier(
    x = c(0, 0.5, 1),
    y = c(0, 1, 0)
  ),
  col = "red",
  lwd = 2
)
```

boxedText *Add Text in a Box to a Plot*

## Description

boxedText draws the strings given in the vector labels at the coordinates given by x and y, surrounded by a rectangle.

## Usage

```
boxedText(x, ...)

## Default S3 method:
boxedText(
  x,
  y = NULL,
  labels = NULL,
  adj = NULL,
  pos = NULL,
  offset = 0.5,
  vfont = NULL,
  cex = 1,
  col = NULL,
  font = NULL,
  srt = 0,
  xPad = 0.2,
  yPad = 0.2,
  density = NULL,
  angle = 45,
  bg = NA,
  border = par("fg"),
  lty = par("lty"),
  lwd = par("lwd"),
  ...
)

## S3 method for class 'formula'
boxedText(formula, data = parent.frame(), ..., subset)
```

## Arguments

| | |
|---|---|
| x, y | numeric vectors of coordinates where the text labels should be written. If the length of x and y differs, the shorter one is recycled. |
| ... | additional arguments are passed to the text function. |
| labels | a character vector or expression specifying the text to be written. An attempt is made to coerce other language objects (names and calls) to expressions, and |

| | |
|---|---|
| | vectors and other classed objects to character vectors by as.character. If labels is longer than x and y, the coordinates are recycled to the length of labels. |
| adj | the value of adj determines the way in which text strings are justified. A value of 0 produces left-justified text, 0.5 (the default) centered text and 1 right-justified text. (Any value in `[0, 1]` is allowed, and on most devices values outside that interval will also work.) Note that the adj argument of text also allows adj = c(x, y) for different adjustment in x- and y- directions. |
| pos | a position specifier for the text. If specified this overrides any adj value given. Values of 1, 2, 3 and 4, respectively indicate positions below, to the left of, above and to the right of the specified coordinates. |
| offset | when pos is specified, this value gives the offset of the label from the specified coordinate in fractions of a character width. |
| vfont | `NULL` for the current font family, or a character vector of length 2 for Hershey vector fonts. The first element of the vector selects a typeface and the second element selects a style. Ignored if labels is an expression. |
| cex | numeric character expansion factor; multiplied by `par("cex")` yields the final character size. `NULL` and `NA` are equivalent to 1.0. |
| col, font | the color and (if vfont = NULL) font to be used, possibly vectors. These default to the values of the global graphical parameters in `par()`. |
| srt | the string rotation in degrees. |
| xPad, yPad | the proportion of the rectangles to the extent of the text within. |
| density | the density of shading lines, in lines per inch. The default value of `NULL` means that no shading lines are drawn. A zero value of density means no shading lines whereas negative values (and NA) suppress shading (and so allow color filling). |
| angle | angle (in degrees) of the shading lines. |
| bg | color(s) to fill or shade the rectangle(s) with. The default `NA` (or also NULL) means do not fill, i.e., draw transparent rectangles, unless density is specified. |
| border | color for rectangle border(s). The default is `par("fg")`. Use `border = NA` to omit borders (this is the default). If there are shading lines, `border = TRUE` means use the same colour for the border as for the shading lines. |
| lty | line type for borders and shading; defaults to `"solid"`. |
| lwd | line width for borders and shading. Note that the use of `lwd = 0` (as in the examples) is device-dependent. |
| formula | A formula of the form `lhs ~ rhs`, where `lhs` gives the response values and `rhs` the corresponding groups or explanatory variables. |
| data | an optional matrix or data frame (or similar; see `stats::model.frame()`) containing the variables in the formula. By default the variables are taken from `environment(formula)`. |
| subset | an optional vector specifying a subset of observations to be used in the analysis. |

## Value

No return value, called for its side effect of adding the boxed labels to the current plot.

## See Also

similar function in package **plotrix** `plotrix::boxed.labels` (lacking rotation option)

Other graphics.annotation: `barText()`, `colLegend()`, `errBars()`, `stamp()`, `textLegend()`, `titleRect()`

## Examples

```
canvas(xpd=TRUE)

boxedText(0, 0, adj=0, label="This is boxed text", srt=seq(0,360,20),
          xPad=.3, yPad=.3)
points(0,0, pch=15)

plot(mpg ~ hp, data=mtcars, type="n", main="MT cars mpg/hp (log-log)",
     panel.first=quote(grid()), las=1, log="xy")
boxedText(mpg ~ hp, data=mtcars,
          labels=rownames(mtcars), cex=0.6, border=FALSE, bg="grey90")
```

---

# canvas *Canvas for Geometric Plotting*

---

## Description

This is just a wrapper for creating an empty plot with suitable defaults for plotting geometric shapes.

## Usage

```
canvas(xlim = NULL, ylim = xlim, main = NULL, asp = 1, bg = "white", ...)
```

## Arguments

| | |
|---|---|
| `xlim`, `ylim` | the xlims and ylims for the plot. Default is c(-1, 1). |
| `main` | the main title on top of the plot. |
| `asp` | numeric, giving the aspect ratio y/x. (See `plot.window()` for details. Default is 1. |
| `bg` | the color of the user space of the plot, defaults to "white". |
| `...` | additional arguments are passed to the `plot()` command. |

## Details

The plot is created with these settings:
`asp = 1, xaxt = "n", yaxt = "n", xlab = "", ylab = "", frame.plot = FALSE`.

## Value

a list of all the previous values of the parameters changed (returned invisibly)

## See Also

Other graphics.setup: polarGrid(), setBackCol()

## Examples

```
canvas(7)
text(0, 0, "Hello world!", cex=5)
```

---

`circle` *Circle Geometry*

---

## Description

Create a circular geometry.

## Usage

```
circle(x = 0, y = 0, radius = 1, nPoints = 100)
```

## Arguments

| | |
|---|---|
| x, y | centre coordinates. |
| radius | circle radius. |
| nPoints | number of points used to approximate the circle. |

## Value

An object inheriting from class "circleGeometry".

## See Also

Other geometry.structures: arc(), band(), bezier(), ellipse(), polygonX(), regPolygon(), ring()

## Examples

```
canvas(xlim = c(-2, 2))

polygonX(circle(radius = 1.5), col = "lightblue")

# several circles at once
polygonX(circle(x = c(-0.6, 0.6), y = 0, radius = 0.4),
         col = c("tomato", "royalblue"))
```

## cmykToCmy *Convert CMYK to CMY*

### Description

Convert CMYK colors to CMY.

### Usage

```
cmykToCmy(col)
```

### Arguments

| | |
|---|---|
| col | numeric CMYK matrix. |

### Value

Numeric CMY matrix.

### See Also

color-conversion-overview

### Examples

```
cmy <- rgbToCmy(c("#FF6347", "#4169E1"), maxColorValue = 255)
cmyk <- cmyToCmyk(cmy)

# the way back
cmykToCmy(cmyk)
```

## cmykToRgb *Convert CMYK to RGB*

### Description

Convert CMYK colors to RGB.

### Usage

```
cmykToRgb(col, maxColorValue = 1)
```

### Arguments

| | |
|---|---|
| col | numeric CMYK matrix (columns C, M, Y, K). |
| maxColorValue | maximum channel value. |

## Value

Numeric RGB matrix.

## See Also

color-conversion-overview

## Examples

```
cmyk <- cmyToCmyk(rgbToCmy(c("#FF6347", "#4169E1"), maxColorValue = 255))
cmykToRgb(cmyk)

# back to hexadecimal colors
rgbToHex(t(cmykToRgb(cmyk) * 255))
```

---

# cmyToCmyk *Convert CMY to CMYK*

---

## Description

Convert CMY colors to CMYK.

## Usage

```
cmyToCmyk(col)
```

## Arguments

| | |
|---|---|
| col | numeric CMY matrix. |

## Value

Numeric CMYK matrix.

## See Also

color-conversion-overview

## Examples

```
cmy <- rgbToCmy(c("#FF6347", "#4169E1"), maxColorValue = 255)
cmyToCmyk(cmy)
```

`colLegend` *Add a Color Legend to a Plot*

## Description

Draw a color legend (color strip) inside an existing plot region.

## Usage

```
colLegend(
  x,
  y = NULL,
  col = rev(heat.colors(100)),
  labels = NULL,
  title = NULL,
  width = NULL,
  height = NULL,
  horiz = FALSE,
  xjust = 0,
  yjust = 1,
  inset = 0,
  region = c("plot", "figure", "device"),
  border = NA,
  box = FALSE,
  adj = NULL,
  cex = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | left x-coordinate of the legend or a keyword specifying automatic placement: `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"`, `"topright"`, `"right"`, `"center"`. |
| `y` | top y-coordinate of the legend when `x` is numeric. |
| `col` | vector of colors. |
| `labels` | optional vector of labels, or a list holding the labels (element `text`, or the one unnamed element) and their placement relative to the color blocks in the element `adj`:<br>`"edge"` (default) Labels are aligned with the strip edges.<br>`"center"` Labels are centered within color blocks.<br>E.g. `labels = list(1:5, adj = "center")`. |
| `title` | optional title, or a list holding the title (element `text`, or the one unnamed element) and its horizontal adjustment in the element `adj` (default 0.5, centered), e.g. `title = list("From black to red", adj = 0)`. |

| | |
|---|---|
| width | width of the legend in user coordinates. |
| height | height of the legend in user coordinates. |
| horiz | logical; if TRUE, draw horizontally. |
| xjust | horizontal justification. |
| yjust | vertical justification. |
| inset | inset distance(s) when keyword positioning is used. |
| region | character string specifying the reference region used for keyword placement. One of: "plot", "figure", or "device". |
| border | border color of individual color rectangles. |
| box | optional specification of an enclosing box around the whole legend. Can be:<br>• FALSE, NULL, or NA: no box<br>• TRUE: draw box with defaults<br>• named list of arguments passed to rect() |
| adj | text alignment passed to text(). |
| cex | character expansion for labels. |
| ... | additional arguments passed to text(). |

## Details

The legend can be positioned either by explicit coordinates or by keyword placement such as "topright" or "left".

Labels may either be aligned with the edges of the color strip or centered within the color blocks.

## Value

Invisibly returns a list with components:

**rect** List describing the color strip geometry with components width, height, left, and top.

**text** List containing the label coordinates x and y.

## See Also

graphics::legend

Other graphics.annotation: barText(), boxedText(), errBars(), stamp(), textLegend(), titleRect()

## Examples

```
plot(1:15,, xlim=c(0,10), type="n", xlab="", ylab="", main="Colorstrips")

# A
colLegend(x="right", inset=5, labels=c(1:10))

# B: Center the labels
colLegend(x=1, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=list(1:5, adj = "center"))
```

```
# C: Outer frame
colLegend(x=3, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:4, box=list(border="grey", lty="dashed"))

# D
colLegend(x=5, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(10), labels=sprintf("%.1f",seq(0,1,0.1)), cex=0.8)

# E: horizontal shape
colLegend(x=1, y=2, width=6, height=0.2, col=rainbow(500), labels=1:5,horiz=TRUE)

# F
colLegend(x=1, y=14, width=6, height=0.5, col=colorRampPalette(
  c("black","blue","green","yellow","red"), space = "rgb")(100),
  horiz=TRUE, box = TRUE)

# G
colLegend(x=1, y=12, width=6, height=1, col=colorRampPalette(c("black","blue",
            "green","yellow","red"), space = "rgb")(10), horiz=TRUE,
            border="black", title=list("From black to red", adj=0))

text(x = c(8,0.5,2.5,4.5,0.5,0.5,0.5)+.2, y=c(14,9,9,9,2,14,12), LETTERS[1:7], cex=2)
```

---

`color-conversion-overview`

*Color Conversion Functions in pharos*

---

## Description

pharos provides a family of functions for converting colors between representations. Read each matrix as **row -> column**: the cell at row X, column Y is the function that converts a color from representation X to representation Y. - marks the diagonal (no self-conversion), . marks a combination with no direct function. RGB is the hub between the two tables below.

### R color, hex, HSV, and RGB

| From \ To | Col | Hex | HSV | RGB |
|---|---|---|---|---|
| **Col** | - | `colToHex()` | `colToHsv()` | `colToRgb()` |
| **Hex** | `hexToCol()` | - | . | `hexToRgb()` |
| **HSV** | . | . | - | . |
| **RGB** | `rgbToCol()` | `rgbToHex()` | . | - |

*"Col" is any valid R color specification (name, hex string, or palette index) as accepted by* `grDevices::col2rgb()`. No function starts from HSV: it is only ever a conversion target (via `colToHsv()`), not a source – see the note below the second table for the reason this gap is left open.

### RGB, CMY, CMYK, and long integer

| From \ To | CMY | CMYK | Long | RGB |
|---|---|---|---|---|
| **CMY** | - | `cmyToCmyk()` | . | . |
| **CMYK** | `cmykToCmy()` | - | . | `cmykToRgb()` |
| **Long** | . | . | - | `longToRgb()` |
| **RGB** | `rgbToCmy()` | . | `rgbToLong()` | - |

### Not part of either conversion matrix

Two related functions transform a color without changing its representation, so they don't fit a row/column slot above:

| Function | Purpose |
|---|---|
| `colToOpaque()` | Computes the equivalent opaque color for a transparent color against a background – Hex stays Hex |
| `grayScale()` | Converts colors to grayscale using luminance weighting – Col stays Col |

### Why HSV has no source functions

Base R already provides the HSV -> Col/Hex direction via `grDevices::hsv()`, which builds a hex color string directly from h/s/v values – the same role `rgb()` plays for RGB triplets. pharos deliberately doesn't duplicate it; chain `hsv()` into `colToRgb()` or `colToHex()` instead (see examples).

### See Also

`grDevices::col2rgb()`, `grDevices::hsv()`

### Examples

```
# A "." in the matrix means chaining two functions through a hub
# (usually RGB or Col), not a missing feature.

# CMY -> RGB: no direct function: CMY -> CMYK -> RGB
c(0.2, 0.6, 0.9) |> cmyToCmyk() |> cmykToRgb()

# Long integer -> R color name: Long -> RGB -> Col
255 |> longToRgb() |> rgbToCol()

# HSV -> RGB: base R's hsv() bridges the gap noted above
hsv(h = 0.6, s = 0.8, v = 0.9) |> colToRgb()
```

## colToHex *Convert R Colors to Hexadecimal Colors*

### Description

Convert any valid R color specification to hexadecimal colors.

### Usage

```
colToHex(col, opacity = 1)
```

### Arguments

| | |
|---|---|
| col | vector of valid R colors. |
| opacity | opacity value between 0 and 1. |

### Value

Character vector of hexadecimal colors.

### See Also

color-conversion-overview

### Examples

```
colToHex(c("tomato", "royalblue"))
colToHex("tomato", opacity = 0.5)

# color numbers refer to the current palette
colToHex(1:3)
```

## colToHsv *Convert R Colors to HSV*

### Description

Convert any valid R color specification to HSV.

### Usage

```
colToHsv(col, useAlphaChannel = FALSE)
```

### Arguments

| | |
|---|---|
| col | vector of valid R colors. |
| useAlphaChannel | logical indicating whether the alpha channel should be included. |

### Value

Numeric HSV matrix with one column per color. With useAlphaChannel = TRUE the alpha channel is added as fourth row, scaled to the range from 0 to 1 like the other rows.

### See Also

color-conversion-overview

### Examples

```
colToHsv(c("tomato", "royalblue"))

# the alpha channel is added as fourth row, on a scale from 0 to 1
colToHsv("#FF634780", useAlphaChannel = TRUE)
```

---

## colToOpaque *Equivalent Opaque Color for Transparent Color*

---

### Description

Determine the equivalent opaque RGB color for a given partially transparent RGB color against a background of any color.

### Usage

```
colToOpaque(col, opacity = NULL, bg = NULL)
```

### Arguments

| | |
|---|---|
| col | the color as hex value (use converters below if it's not available). col and opacity are recycled. |
| opacity | the opacity value, if left to NULL the alpha channels of the colors are used |
| bg | the background color to be used to calculate against (default is "white") |

### Details

Reducing the opacity against a white background is a good way to find usable lighter and less saturated tints of a base color. For doing so, we sometimes need to get the equivalent opaque color for the transparent color.

## Value

An named vector with the hexcodes of the opaque colors.

## See Also

Other color.manipulation: `addOpacity()`, `darken()`, `fade()`, `lighten()`, `mixCol()`

## Examples

```
cols <- c(addOpacity("limegreen", 0.4),
          colToOpaque(colToHex("limegreen"), 0.4),
          "limegreen")
barplot(c(1, 1.2, 1.3), col=cols, panel.first=abline(h=0.4, lwd=10, col="grey35"))
```

---

`colToRgb` *Convert R Colors to RGB*

---

## Description

Convert any valid R color specification to an RGB matrix.

## Usage

```
colToRgb(col, useAlphaChannel = FALSE)
```

## Arguments

| | |
|---|---|
| `col` | vector of valid R colors. |
| `useAlphaChannel` | logical indicating whether the alpha channel should be included. |

## Value

Integer matrix with rows corresponding to RGB (and optionally alpha) channels.

## See Also

grDevices::col2rgb, color-conversion-overview

## Examples

```
rgbToCol(matrix(c(162,42,42), nrow=3))

rgbToLong(matrix(c(162,42,42), nrow=3))

colToRgb("peachpuff")
colToRgb(c(blu = "royalblue", reddish = "tomato")) # names kept

colToRgb(1:8)
```

---

`contrastCol` *Choose Optimal Text Color Based on WCAG Contrast*

---

## Description

Computes the optimal text color (default: black or white) for a given background color based on the WCAG contrast ratio.

## Usage

```
contrastCol(col, light = "white", dark = "black")
```

## Arguments

| | |
|---|---|
| `col` | vector of colors. Can be any valid R color specification: color names, hexadecimal strings (e.g. `"#RRGGBB"` or `"#RRGGBBAA"`), or palette indices. |
| `light` | color used for dark backgrounds (default: `"white"`). |
| `dark` | color used for light backgrounds (default: `"black"`). |

## Details

This function implements the official relative luminance definition for sRGB colors (including gamma correction) and selects the text color that maximizes the contrast ratio according to the Web Content Accessibility Guidelines (WCAG 2.1).

Compared to simple heuristics (e.g., mean RGB), this approach is perceptually more accurate and suitable for accessibility-critical applications.

The relative luminance is computed as:

$$L = 0.2126R + 0.7152G + 0.0722B$$

where $R, G, B$ are linearized sRGB components:

$$c_{lin} = \begin{cases} \frac{c}{12.92} & c \leq 0.04045 \\ \left(\frac{c+0.055}{1.055}\right)^{2.4} & c > 0.04045 \end{cases}$$

The contrast ratio between two luminance values $L1$ and $L2$ is:

$$\frac{\max(L1, L2) + 0.05}{\min(L1, L2) + 0.05}$$

The function compares contrast ratios against `light` and `dark` and returns the better option.

## Value

A character vector of the same length as `col`, containing either `light` or `dark`, depending on which yields the higher contrast ratio.

## See Also

Other color.lookup: `findCol()`

## Examples

```
cols <- c("black", "white", "red", "blue", "yellow", "#777777")
contrastCol(cols)

# custom text colors
contrastCol(cols, light = "#FFFFFF", dark = "#222222")
```

---

# convUnit *Symbolic Unit Conversion Engine*

---

## Description

Converts numeric values between units using a symbolic unit system, SI-derived expansions, and graph-based conversion for non-SI units.

## Usage

```
convUnit(x, from, to, prefix = NULL, units = NULL)
```

## Arguments

| | |
|---|---|
| `x` | numeric value(s) to convert. |
| `from` | character string specifying the source unit. |
| `to` | character string specifying the target unit. |
| `prefix` | data frame of SI prefixes with columns `abbr` and `mult`. |
| `units` | data frame of unit conversion factors with columns `from`, `to`, and `fact`. |

## Details

Supports:

- SI base units (m, kg, s, A, K, mol, cd)
- Derived units (N, Pa, J, W, Hz)
- Prefixes (k, m, µ, etc.)
- Compound units (e.g. `"km/h"`, `"kg*m/s^2"`)
- Unit powers (e.g. `"m2"`, `"s^-1"`)
- Temperature conversion (C, F, K)

The function internally:

1. Parses units into symbolic components
2. Expands derived units (e.g. `N = kg*m/s^2`)
3. Computes dimensional vectors
4. Applies prefix scaling
5. Uses a graph search (Dijkstra) for non-SI conversions

The function checks dimensional consistency before conversion. If units are not compatible, an error is thrown.

For non-SI units (e.g. `"mi"`, `"bar"`), conversion paths are resolved via a graph representation of `Units`.

Temperature conversions are handled separately and do not use multiplicative scaling.

## Value

Numeric value(s) converted to the target unit.

## See Also

Other format: `fm()`, `fmCI()`, `print.Unit()`, `style()`, `unit()`

## Examples

```
convUnit(100, "km/h", "m/s")

convUnit(1, "N", "kg*m/s^2")
convUnit(1, "Pa", "N/m^2")
convUnit(25, "C", "F")
convUnit(1, "ml", "l")
```

coordinate-conversions

*Coordinate Transformations Cartesian/Polar/Spherical*

## Description

Transform cartesian into polar coordinates, resp. to spherical coordinates and vice versa.

## Usage

```
polToCart(r, theta)

cartToPol(x, y)

cartToSph(x, y, z, up = TRUE)

sphToCart(r, theta, phi, up = TRUE)
```

## Arguments

| | |
|---|---|
| r | a vector with the radius of the points. |
| theta | a vector with the angle(s) of the points. |
| x, y, z | vectors with the xy-coordianates to be transformed. |
| up | logical. If set to TRUE (default) theta is measured from x-y plane, else theta is measured from the z-axis. |
| phi | a vector with the angle(s) of the points. |

## Details

Angles are in radians, not degrees (i.e., a right angle is pi/2). Use degToRad() to convert, if you don't wanna do it by yourself.
All parameters are recycled if necessary.

## Value

polToCart() returns a list of x and y coordinates of the points.
cartToPol() returns a list of r for the radius and theta for the angles of the given points.

## Note

Based on code by Christian W. Hoffmann

## See Also

Other geometry.conversion: degree-radians-conversion

### Examples

```
cartToPol(x=1, y=1)
cartToPol(x=c(1,2,3), y=c(1,1,1))
cartToPol(x=c(1,2,3), y=1)


polToCart(r=1, theta=pi/2)
polToCart(r=c(1,2,3), theta=pi/2)

polToCart(r=c(1,2,3), theta=pi/2)

cartToSph(x=1, y=2, z=3)   # r=3.741657, theta=0.930274, phi=1.107149
```

---

## darken *Darken Colors*

---

### Description

Darken colors by mixing them with black.

### Usage

```
darken(col, amount = 0.2)
```

### Arguments

| | |
|---|---|
| col | vector of valid R colors. |
| amount | numeric value between 0 and 1 specifying the amount of darkening. A value of 0 leaves the color unchanged, while 1 returns black. |

### Details

Colors are mixed linearly with black in RGB space:

$$x_{new} = x \cdot (1 - amount)$$

### Value

Character vector of hexadecimal colors.

### See Also

Other color.manipulation: `addOpacity()`, `colToOpaque()`, `fade()`, `lighten()`, `mixCol()`

## Examples

```
darken("tomato")
darken(c("tomato", "royalblue"), amount = 0.5)

# increasing amounts
amount <- seq(0, 1, 0.2)
barplot(rep(1, 6), col = sapply(amount, darken, col = "tomato"),
        names.arg = amount, yaxt = "n", xlab = "amount")
```

---

# degree-radians-conversion

*Convert Degrees to Radians and Vice Versa*

---

## Description

Convert degrees to radians (and back again).

## Usage

```
degToRad(deg)

radToDeg(rad)
```

## Arguments

| | |
|---|---|
| deg | a vector of angles in degrees. |
| rad | a vector of angles in radians. |

## Value

degToRad returns a vector of the same length as deg with the angles in radians.
radToDeg returns a vector of the same length as rad with the angles in degrees.

## See Also

Other geometry.conversion: coordinate-conversions

## Examples

```
degToRad(c(90,180,270))
radToDeg( c(0.5,1,2) * pi)
```

`ellipse` *Ellipse Geometry*

## Description

Create an elliptic geometry.

## Usage

```
ellipse(x = 0, y = 0, radiusX = 1, radiusY = radiusX, nPoints = 100)
```

## Arguments

| | |
|---|---|
| `x`, `y` | centre coordinates. |
| `radiusX`, `radiusY` | horizontal and vertical radius. |
| `nPoints` | number of points used to approximate the ellipse. |

## Details

Use `rotate()` to rotate the resulting geometry.

## Value

An object inheriting from class `"ellipseGeometry"`.

## See Also

Other geometry.structures: `arc()`, `band()`, `bezier()`, `circle()`, `polygonX()`, `regPolygon()`, `ring()`

## Examples

```
canvas(xlim = c(-2, 2))

e <- ellipse(radiusX = 1.8, radiusY = 0.8)
polygonX(e, col = "lightblue")

# the same ellipse, rotated by 45 degrees
polygonX(rotate(e, theta = pi/4), border = "red")
```

## embedFile *Base64-encode a file*

### Description

The building block behind `asImg()` and `asFileLink()`: reads a file and returns its contents base64-encoded, ready to be placed in a data URI or in any container format that carries binary payloads as text.

### Usage

```
embedFile(path)
```

### Arguments

| | |
|---|---|
| `path` | path to an existing file |

### Value

a single character string

### See Also

Other html: `asFileLink()`, `asHtml()`, `asImg()`, `escapeHtml()`, `htmlNotation`, `htmlSubscript`, `preview()`, `preview.html()`, `toHtmlTable()`

### Examples

```
fn <- tempfile(fileext = ".txt")
writeLines("hello", fn)
substr(embedFile(fn), 1, 8)
unlink(fn)
```

## errBars *Add Error Bars to an Existing Plot*

### Description

Add vertical or horizontal error bars to an existing plot.

### Usage

```
errBars(from, to = NULL, pos = NULL, horiz = TRUE, points = NULL, ...)
```

## Arguments

| | |
|---|---|
| from | coordinates of the lower end of the error bars.<br>If to = NULL and from is a matrix:<br>• a 2-column matrix is interpreted as cbind(from, to)<br>• a 3-column matrix is interpreted as cbind(point, from, to) |
| to | coordinates of the upper end of the error bars. |
| pos | position of the error bars.<br>For vertical bars this corresponds to x-positions; for horizontal bars to y-positions. |
| horiz | logical; if TRUE, horizontal error bars are drawn. |
| points | optional point specification. |
| ... | additional graphical arguments passed to arrows(). |

## Details

This is a lightweight wrapper around arrows() with optional point symbols added via points().

Additional graphical arguments in ... are passed to arrows() and may be used to control the appearance of the error bars.

Common examples include:

- col: line color
- lwd: line width
- lty: line type
- code: which end caps to draw
- length: length of the end caps

Point symbols may optionally be added:

- points = TRUE: draw default points halfway between from and to
- points = numeric: use these values as point coordinates
- points = list(...): fully specify point coordinates and graphical parameters

Example:

```
points = list(
  val = estimate,
  pch = 21,
  bg  = "gold",
  col = "black",
  cex = 1.2
)
```

The default orientation is horizontal (horiz = TRUE), which suits the typical use case of adding confidence intervals to a dotchart(). Set horiz = FALSE for vertical bars on barplots or similar.

## Value

Invisibly returns a list with components:

**from** Lower endpoints

**to** Upper endpoints

**pos** Positions of the error bars

**points** Point coordinates

## See Also

arrows(), points()

Other graphics.annotation: barText(), boxedText(), colLegend(), stamp(), textLegend(), titleRect()

## Examples

```
op <- par(no.readonly = TRUE)
par(mfrow = c(2, 2))

b <- barplot(1:5, ylim = c(0, 6))

errBars(
  from = 1:5 - 0.5,
  to = 1:5 + 0.5,
  pos = b,
  length = 0.15
)

# midpoint symbols
b <- barplot(1:5, ylim = c(0, 6))

errBars(
  from = 1:5 - 0.5,
  to = 1:5 + 0.5,
  pos = b,
  points = TRUE
)

# custom points
dotchart(1:5, xlim = c(0, 6))

errBars(
  from = 1:5 - 0.2,
  to = 1:5 + 0.2,
  horiz = TRUE,
  points = list(
    val = 1:5,
    pch = 21,
    bg = "gold",
    col = "black",
```

```
    cex = 1.2
  )
)

par(op)
```

---

`escapeHtml` *Escape HTML special characters*

---

## Description

Replaces the three characters that would otherwise be read as markup. Use it on any text of unknown origin – a variable label, a file name, a user-supplied caption – before pasting it into HTML or XML.

## Usage

```
escapeHtml(x, attribute = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a character vector |
| `attribute` | logical; also escape single and double quotes |

## Details

Quotes are escaped as well when `attribute = TRUE`, which is required for text placed inside an attribute value rather than between tags.

## Value

a character vector of the same length

## See Also

Other html: `asFileLink()`, `asHtml()`, `asImg()`, `embedFile()`, `htmlNotation`, `htmlSubscript`, `preview()`, `preview.html()`, `toHtmlTable()`

## Examples

```
escapeHtml("Anteil < 5% & steigend")
escapeHtml('say "hi"', attribute = TRUE)
```

`exportPlot` *Export a Plot to Graphics Files*

## Description

Draws a plot to one or several graphics files (PNG, TIFF, PDF, SVG) in a single call, controlled by a session-wide switch. It is meant for analysis scripts that produce publication figures: the plot call is written once, piped into `exportPlot()`, and the files are only (re)written when the switch is on. Running a chunk interactively then shows the plot on screen without overwriting the figures produced by an earlier, final run.

```
options(pharos.exportPlot = TRUE)   # once, at the top of the script

plotXY(y ~ x, data = d) |>
  exportPlot("fig1", type = c("png", "tif"))
```

## Usage

```
exportPlot(
  expr,
  file,
  type = "png",
  width = 16/3,
  height = width,
  res = 300,
  create = getOption("pharos.exportPlot", FALSE),
  screen = TRUE
)
```

## Arguments

| | |
|---|---|
| `expr` | a call that draws a plot, typically piped in with `\|>`, or a braced block of drawing calls. It is not evaluated before `exportPlot()` runs, and is evaluated several times (see Details). |
| `file` | the path of the output file, without extension; one file per element of `type` is written, with the extension of that type. A trailing extension `.png`, `.tif`, `.tiff`, `.pdf` or `.svg` is removed. Required if `create = TRUE`. |
| `type` | one or several of `"png"`, `"tif"`, `"pdf"` and `"svg"`. Default is `"png"`. TIFF files are LZW-compressed; SVG needs a working cairo device. `capabilities("cairo")` is not enough to tell: on macOS it can be `TRUE` while `svg()` cannot open a device (XQuartz missing). A device that cannot be opened is an error. |
| `width, height` | the size of the plot in inches. The default width of `16/3` inches gives 1600 pixels at 300 dpi; `height` defaults to `width`. |
| `res` | the resolution in dpi for the bitmap formats PNG and TIFF. Ignored for PDF and SVG. |

| | |
|---|---|
| `create` | logical; if `TRUE`, the files are written. Defaults to the option `pharos.exportPlot`, or `FALSE` if the option is not set. |
| `screen` | logical; if `TRUE` (default), the plot is also drawn on the current device when files are written. Without files (`create = FALSE`) the plot is always drawn on the current device. |

## Details

`exportPlot()` does not receive a finished plot but the plot *call*. The native pipe rewrites `f(x) |> exportPlot()` into `exportPlot(f(x))`, and since R evaluates arguments lazily, `f(x)` has not been run when `exportPlot()` starts. The call is captured with `substitute()` and evaluated once per output file on a freshly opened device, and finally once on the current device. Every file is therefore drawn at its own size and resolution: margins, label widths and font sizes that a plot function computes from the device (e.g. via `strwidth()`) are right for each format. Replaying a recorded plot (`recordPlot()`) would freeze the values computed on the screen device instead.

This only works with the native pipe `|>` or with a direct call `exportPlot(plot(...), ...)`, which also accepts a braced block of several drawing calls (see Examples). The magrittr pipe `%>%` hands over a placeholder, not the call.

**Repeated evaluation.** Everything in `expr` runs once per output file and once more on the screen if requested (`screen`): assignments, messages, file operations and expensive computations included. Prepare the data beforehand and pass only the drawing calls.

**Random numbers.** If a random number generator state exists, it is restored before each evaluation, so every output starts from the same state, and the state after the call is that of a single evaluation. A plot that draws random numbers (jitter, sampling) then comes out the same in every file, as long as it consumes the same random numbers on every device; a plot that branches on device properties may not. In a fresh session in which no random number has been drawn yet there is no state to restore, and the outputs are not synchronised; call `set.seed()` first.

**Graphics systems.** Base graphics are drawn by the call itself. `ggplot` and `trellis` objects, which are only drawn when printed, are printed on each device; a grob is drawn on a new page with `grid::grid.draw()`. This applies only to a *visible* result, as at the console: `exportPlot(p, ...)` draws the ggplot `p`, while `exportPlot(print(p), ...)` draws it once through `print()`, which returns it invisibly. An object returned invisibly is never drawn automatically.

**Switch.** `create` defaults to the option `pharos.exportPlot`, which is `FALSE` unless set. A script thus decides in one place whether its figures are written, while single calls can still override it.

**Devices.** Each file device is closed by its own number after drawing, also if the plot call fails or opens or activates another device. The device that was current before the call is made current again, provided it is still open. If a file device cannot be opened, `exportPlot()` stops before drawing, with the device's own message as the reason; the current device is left untouched, and files of earlier types in `type` remain written.

## Value

the paths of the written files as a character vector, invisibly; `character(0)` if `create = FALSE`.

## See Also

`stamp()` for the corner stamp, which plot functions in pharos draw by default and which is usually suppressed (`stamp = FALSE`) in figures for publication.

## Examples

```
# without the switch, the plot is only drawn
plot(1:10) |>
  exportPlot(file.path(tempdir(), "demo"))

# write a PNG and a PDF, nothing on the screen
f <- plot(1:10, main = "Demo") |>
  exportPlot(file.path(tempdir(), "demo"), type = c("png", "pdf"),
             width = 4, create = TRUE, screen = FALSE)
f
file.exists(f)
unlink(f)

# a composed base plot: the model is fitted outside, as expr is
# evaluated once per file
fit <- lm(dist ~ speed, data = cars)
f <- exportPlot({
  plot(dist ~ speed, data = cars)
  abline(fit)
  legend("topleft", legend = "Regression", lty = 1)
}, file.path(tempdir(), "figure1"), create = TRUE, screen = FALSE)
unlink(f)

# switch it on for the whole script
opt <- options(pharos.exportPlot = TRUE)
f <- plot(1:10) |>
  exportPlot(file.path(tempdir(), "demo"), screen = FALSE)
unlink(f)
options(opt)
```

---

`fade` *Fade Colors*

---

## Description

Apply transparency and remove the alpha channel afterwards.

## Usage

```
fade(col, opacity = 0.5)
```

## Arguments

| | |
|---|---|
| `col` | vector of valid R colors |
| `opacity` | opacity value between 0 and 1 |

## Value

Character vector of colors.

## See Also

Other color.manipulation: addOpacity(), colToOpaque(), darken(), lighten(), mixCol()

## Examples

```
fade("tomato")
fade(c("tomato", "royalblue"), opacity = 0.2)

# same tint, but the faded color hides what lies behind it
cols <- c(addOpacity("tomato", 0.4), fade("tomato", 0.4))
barplot(c(1, 1), col = cols,
        panel.first = abline(h = 0.5, lwd = 10, col = "grey35"))
```

---

# findCol *Get Color on a Defined Color Range*

---

## Description

Find a color on a defined color range depending on the value of x. This is helpful for colorcoding numeric values.

## Usage

```
findCol(
  x,
  col = rev(heat.colors(100)),
  minX = NULL,
  maxX = NULL,
  all.inside = FALSE
)
```

## Arguments

| | |
|---|---|
| x | numeric. |
| col | a vector of colors. |
| minX | the x-value to be used for the left edge of the first color. If left to the default NULL min(pretty(x)) will be used. |
| maxX | the x-value to be used for the right edge of the last color. If left to the default NULL max(pretty(x)) will be used. |
| all.inside | logical; if true, the returned indices are coerced into 1, ..., N-1, i.e., 0 is mapped to 1 and N to N-1. |

## Details

For the selection of colors the option `rightmost.closed` in the used function `findInterval()` is set to TRUE. This will ensure that all values on the right edge of the range are assigned a color. How values outside the boundaries of minX and maxX should be handled can be controlled by `all.inside`. Set this value to TRUE, if those values should get the colors at the edges or set it to FALSE, if they should remain white (which is the default).

Note that `findInterval()` closes the intervals on the left side, e.g. [0, 1). This option can't be changed. Consequently will x-values lying on the edge of two colors get the color of the bigger one.

## Value

A vector of the length of `x` with the colors assigned to the values, taken from `col` (names of `col` are kept). Values outside the range `minX` to `maxX` yield `NA`, unless `all.inside` is `TRUE`.

## See Also

`findInterval()`

Other color.lookup: `contrastCol()`

## Examples

```
canvas(7, main="Use of function findCol()")

# get some data
x <- c(23, 56, 96)
# get a color range from blue via white to red
cols <- colorRampPalette(c("blue","white","red"))(100)
colLegend(x="bottomleft", col=cols, labels=seq(0, 100, 10), cex=0.8)

# and now the color coding of x:
(xcols <- findCol(x, cols, minX=0, maxX=100))

# this should be the same as
cols[x+1]

# how does it look like?
y0 <- c(-5, -2, 1)
text(x=1, y=max(y0)+2, labels="Color coding of x:")
text(x=1.5, y=y0, labels=x)
polygonX(regPolygon(x=3, y=y0, nVertices=4, startAngle=pi/4), col=xcols)
text(x=6, y=y0, labels=xcols)

# how does the function select colors?
canvas(xlim = c(0,1), ylim = c(0,1))
cols <- c(red="red", yellow="yellow", green="green", blue="blue")
colLegend(x=0, y=1, width=1, col=rev(cols), horiz = TRUE,
            labels=format(seq(0, 1, .25), digits=2, nsmall=2),
            box=list(border="grey"), cex=0.8 )
```

```
x <- c(-0.2, 0, 0.15, 0.55, .75, 1, 1.3)
arrows(x0 = x, y0 = 0.6, y1 = 0.8, angle = 15, length = .2)
text(x=x, y = 0.5, labels = x, adj = c(0.5,0.5))
text(x=x, y = 0.4, labels = names(findCol(x, col=cols,
   minX = 0, maxX = 1, all.inside = TRUE)), adj = c(0.5,0.5))
text(x=x, y = 0.3, labels = names(findCol(x, col=cols,
   minX = 0, maxX = 1, all.inside = FALSE)), adj = c(0.5,0.5))
```

---

`fm` *Format Numbers and Dates*

---

## Description

Formatting numbers with base R tools often degenerates into a major intellectual challenge for us little minds down here in the valley of tears. There are a number of options available and quite often it's hard to work out which one to use, when a more uncommon setting is needed. The `fm()` function wraps all these functions and tries to offer a simpler, less technical, but still flexible interface.

## Usage

```
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)

## Default S3 method:
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
```

```
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)

## S3 method for class 'data.frame'
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)

## S3 method for class 'matrix'
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)

## S3 method for class 'table'
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
```

```
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)

## S3 method for class 'ftable'
fm(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a numeric, logical, character, factor, Date, or POSIXt vector, or a matrix, table, ftable, or data frame |
| digits | integer, the desired (fixed) number of digits after the decimal point. You will always get this number of digits, even if the last digit is 0. In scientific notation it is the number of decimals of the mantissa. Negative numbers of digits round to a power of ten (digits=-2 would round to the nearest hundred) for standard numeric formats; engineering formats require nonnegative values |
| leadDigits | number of leading zeros. leadDigits=3 would make sure that at least 3 digits on the left side will be printed, say 3.4 will be printed as 003.4. Setting leadDigits to 0 will yield results like .452 for 0.452. The default NULL will leave the numbers as they are (meaning at least one 0 digit). |
| sci | numeric scalar giving the absolute power-of-ten threshold for scientific notation. It applies to the absolute value of x and is used symmetrically: for sci = 8, a value is displayed scientifically if $\|x\| \geq 10^8$ or $0 < \|x\| < 10^{-8}$. The default is based on getOption("scipen"); an option value of zero is replaced by 7 |

| | |
|---|---|
| bigMark | character; if not empty used as mark between every 3 digits before the decimal point. If NULL, the option bigMark is used and, if that is not set, "" (none). |
| decMark | character specifying the decimal mark. If NULL, the current OutDec option is used |
| naForm | character, string specifying how NAs should be specially formatted. If set to NULL (default) no special action will be taken. |
| zeroForm | character, string specifying how zeros should be specially formatted. Useful for pretty printing 'sparse' objects. If set to NULL (default) no special action will be taken. |
| fmt | a format code or date-time template, a formatting function, a named Style, or an object of class Style. See Details |
| pThreshold | positive numeric threshold below which p-values are shown as "< threshold" |
| width | nonnegative integer giving the minimum display width |
| align | the character on whose position the strings will be aligned. Left alignment can be requested by setting align = "\\l", right alignment by "\\r" and center alignment by "\\c". Mind the backslashes, as if they are omitted, strings would be aligned to the **character** l, r or c respectively. The default is NULL which would just leave the strings as they are.<br>This argument is send directly to the function strAlign() as argument sep. |
| lang | optional value setting the language for the months and daynames. Can be either "local" for current locale or "en" for english. If left to NULL, the option DescToolsX.lang is used, falling back to "en" |
| ... | additional arguments passed to methods or to a formatting function supplied through fmt |

## Details

There's also an easygoing interface for format templates, defined as a list consisting of any accepted format features. This enables to define templates globally and easily change or modify them later.

fm() is the workhorse for formatting numbers and dates, supporting a comprehensive range of format options that are likely to occur in everyday reporting. Among these, the argument fmt deserves a more detailed description due to its flexibility. It is used to generate a variety of different special formats.

If x is a date, it can take ISO-8601-inspired token syntax similar to .NET or Moment.js (consisting of d, M and y for day, month or year and h/H, m, s, t for hours, minutes, seconds and AM/PM designator) and defining the combination of day month and year representation.

| Code | Description |
|---|---|
| d | day of the month without leading zero (1 - 31) |
| dd | day of the month with leading zero (01 - 31) |
| ddd | abbreviated name for the day of the week (e.g. Mon) in the current user's language |
| dddd | full name for the day of the week (e.g. Monday) in the current user's language |
| do | The token do (aka 'day ordinal') formats the day of month using English ordinal suffixes (e.g. 1st, 2nd, 3rd, 4th). Thi |
| M | month without leading zero (1 - 12) |

| | |
|---|---|
| MM | month with leading zero (01 - 12) |
| MMM | abbreviated month name (e.g. Jan) in the current user's language |
| MMMM | full month name (e.g. January) in the current user's language |
| y | year without century, without leading zero (0 - 99) |
| yy | year without century, with leading zero (00 - 99) |
| yyyy | year with century. For example: 2005 |

| | |
|---|---|
| H/HH | Hour in 24h format, one digit / two digits |
| h/hh | Hour in 12h format, one digit / two digits, note that in this case t must be set also to ensure uniqueness. |
| t/tt | AM/PM description (one/two characters) |
| m/mm | Minutes one digit / two digits |
| s/ss | Seconds one digit / two digits |

Weekdays and month names can be expressed in the local language or in English. The language can be controlled by the argument "lang".

Even more variability is needed to display numeric values. For the most frequently used formats there are the following special codes available:

| Code | Type | Description |
|---|---|---|
| e | scientific | forces scientific representation of x, e.g. 3.141e-05. digits gives the number of decimals of the mantissa; alignment and zero values are further respected. |
| eng | engineering | forces scientific representation of x, but only with powers that are a multiple of 3. |
| engabb | engineering abbr. | same as eng, but replaces the exponential representation by codes, e.g. M for mega (1e6). |
| \% | percent | multiplies the given number by 100 and appends the \ formats values as p-values. Use pThreshold to define the threshold to e.g. switch to a <0.001 representation. |
| frac | fractions | will (try to) convert numbers to fractions. So 0.1 will be displayed as 1/10. See MASS::fractions(). |
| * | significance | will produce a significance representation of a p-value consisting of * and ., while the breaks are set according to the used defaults e.g. in lm as [0, 0.001] = *** (0.001, 0.01] = ** (0.01, 0.05] = * (0.05, 0.1] = . (0.1,1] = |
| p* | p-value stars | will produce p-value and significance stars |

fmt can as well be an object of class "Style" consisting of a list out of the arguments above (as created by style()). This allows to store and manage the full format in variables or as options and use it as format template subsequently. Arguments supplied directly to fm() override the corresponding Style settings. NULL means "not specified" and therefore leaves the Style setting in place.

For data frames and ftables, every formatting argument must have length one or the number of columns. Length-one arguments are recycled, allowing each column to use its own formatting settings without ambiguous partial recycling. Functions and Style objects count as single settings; use a list to supply different functions or Styles by column.

Finally, `fmt` can be a function of `x`. Additional arguments in `\dots` are forwarded to that function.

## Value

formatted character values with the dimensions or tabular structure of `x` preserved where applicable, of class `noquote` so that they print without quotation marks. Every value is formatted on its own, whatever the shape of `x`: a matrix or a table is formatted exactly like the vector of its entries and is not padded to a common width. Padding is applied only on request, via `width` or `align`. To display a formatted matrix with its entries lined up, print it right justified, e.g. `print(fm(m), right = TRUE)`, or align it explicitly.

## See Also

base::format, base::formatC, base::prettyNum, base::sprintf, stats::symnum, base::Sys.setlocale, `DescToolsX::weekday`, `DescToolsX::month`, theme

Other format: `convUnit()`, `fmCI()`, `print.Unit()`, `style()`, `unit()`

## Examples

```
# mind the case: M is the month, m the minutes
fm(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d MMMM yyyy")
fm(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d MMMM yyyy", lang="local")

# using english ordinal suffixes
fm(as.Date("2026-01-21"), fmt="MMMM do yyyy", lang="en")
# e.g. in context:
gettextf("Report generated on %s",
         fm(as.Date("2026-05-04"), fmt="MMMM do, yyyy", lang="en"))

# numeric formats
x <- pi * 10^(-10:10)

fm(x, digits=3, fmt="%")
fm(x, digits=4, sci=4, leadDigits=0, width=9, align=".")


# format a matrix: every entry on its own, as for a vector ...
m <- matrix(runif(100) - 0.5, nrow=10,
            dimnames=list(LETTERS[1:10], LETTERS[1:10]))

fm(m, digits=1)
# ... lined up for display only when asked for
print(fm(m, digits=1), right = TRUE)
fm(m, digits=1, align=".")

# engineering format
```

```
fm(x, fmt="eng",  digits=2)
fm(x, fmt="engabb", leadDigits=2, digits=2)
# combine with grams [g]
paste(fm(x, fmt="engabb", leadDigits=2, digits=2), "g", sep="")

# example form symnum
pval <- rev(sort(c(outer(1:6, 10^-(1:3)))))
noquote(cbind(fm(pval, fmt="p"), fm(pval, fmt="*")))

# change the character to be used as the decimal point
fm(1200, digits=2, bigMark = ".", decMark=",")
```

---

`fmCI` *Format Confidence Intervals*

---

## Description

Format confidence intervals using a flexible template.

## Usage

```
fmCI(x, template = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | the numerical values, given in the order in which they are used in the template. |
| `template` | character string as template for the desired format. %s are the placeholders for the numerical values. Default is `` `"%s [%s, %s]"` `` for `<est> [<lci>, <uci>]`. |
| `...` | the dots are passed on to the `fm()` function. |

## Value

a formatted string

## See Also

`fm()`

Other format: `convUnit()`, `fm()`, `print.Unit()`, `style()`, `unit()`

## Examples

```
x <- c(est=2.1, lci=1.5, uci=3.8)

# default template
fmCI(x)
# user defined template (note the double percent code)
fmCI(x, template="%s (95%%-CI %s-%s)", digits=1)
```

```
# in higher datastructures
tapply(warpbreaks$breaks, warpbreaks$wool,
       function(x) fmCI(c(mean(x), t.test(x)$conf.int),
                        digits=1))
```

---

`ftable.list` *Flat Contingency Table for tapply-Like Lists*

---

## Description

Creates a flat contingency table from a list array, such as the result of `tapply()` when the applied function returns a named vector.

## Usage

```
## S3 method for class 'list'
ftable(x, row.vars = NULL, col.vars = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | A list with a `dim` attribute, typically produced by `tapply()`. Each element must be a vector of equal length and have identical names. |
| `row.vars` | row variables passed to `ftable()`. Defaults to all dimensions except those specified in `col.vars`. |
| `col.vars` | column variables passed to `ftable()`. Defaults to the dimension created from the names of the list elements. |
| `...` | further arguments passed to `ftable()`. |

## Details

Each list element is expanded into an additional dimension corresponding to the names of the returned vector. The resulting array is then passed to `ftable()`.

This is particularly useful for displaying multi-dimensional summaries such as confidence intervals returned by `meanCI()`, where each cell contains several statistics (e.g. estimate, lower CI, upper CI).

The names of the vectors stored in the list elements become an additional dimension of the resulting array. By default, this new dimension is shown in the columns of the flat contingency table.

## Value

An object of class `"ftable"`.

## See Also

`tapply()`, `ftable()`

## Examples

```
# a function returning a named vector, here the mean and its
# confidence interval (as e.g. lumen::meanCI() returns them)
meanCI <- function(z) {
  ci <- t.test(z)$conf.int
  c(est = mean(z), lci = ci[1], uci = ci[2])
}

x <- with(
  warpbreaks,
  tapply(
    breaks,
    list(wool, tension),
    meanCI
  )
)

# the statistics form the columns
ftable(x)

# the second grouping variable in the columns as well
ftable(
  x,
  row.vars = 2,
  col.vars = c(3, 1)
)
```

---

`grayScale` *Convert Colors to grayScale*

---

## Description

Convert colors to grayScale using luminance weighting.

## Usage

```
grayScale(col)
```

## Arguments

| | |
|---|---|
| `col` | vector of valid R colors. |

## Details

grayScale conversion uses the standard luminance approximation:

$$0.3R + 0.59G + 0.11B$$

## Value

Character vector of grayScale colors.

## See Also

color-conversion-overview

## Examples

```
op <- par(no.readonly = TRUE)
par(mfcol=c(2,2))

tmp <- 1:3
names(tmp) <- c('red','green','blue')

barplot(tmp, col=c('red','green','blue'))
barplot(tmp, col=grayScale(c('red','green','blue')))

barplot(tmp, col=c('red','#008100','#3636ff'))
barplot(tmp, col=grayScale(c('red','#008100','#3636ff')))

par(op)
```

---

# hcol *Helsana Colors*

---

## Description

Retrieve one or more colors from the helsana palette, optionally faded.

## Usage

```
hcol(..., fade = NULL)
```

## Arguments

| | |
|---|---|
| `...` | character strings naming the colors to retrieve. Valid names are: `"blue"`, `"red"`, `"orange"`, `"yellow"`, `"ecru"`, `"green"`, `"pink"`, `"moss"`, `"slate"`, `"sand"`, `"brown"`, `"plum"`. If none are provided, the full palette is returned. |
| `fade` | `NULL` (default) returns the colors unchanged. A number in $[0, 1]$ returns the opaque tint the color has at this opacity on a white background, see `fade()`: `1` is the color itself, smaller values are lighter. Recycled over the colors, so `hcol("blue", fade = c(1, 0.6, 0.3))` gives three tints of one color. |

## Value

A named character vector of hex color codes.

## See Also

Other color.palettes: pal(), palNames()

## Examples

```
hcol("blue", "green")
hcol()

# lighter tints, still opaque
hcol("blue", "red", fade = 0.4)
hcol("blue", fade = c(1, 0.6, 0.3))
```

---

# hexToCol *Convert Hex Colors to Named R Colors*

---

## Description

Convert hexadecimal colors to the nearest named R colors.

## Usage

```
hexToCol(col, method = c("ciede2000", "cie76"))
```

## Arguments

| | |
|---|---|
| col | character vector of hexadecimal colors. |
| method | color difference formula, "ciede2000" (default) or "cie76", see Details. |

## Value

Character vector of named R colors.

## See Also

color-conversion-overview

## Examples

```
hexToCol(c("#A22A2A", "#4169E0"))
hexToCol("#4169E0", method = "cie76")
```

## hexToRgb *Convert Hex Colors to RGB*

### Description

Convert hexadecimal color strings to an RGB matrix.

### Usage

```
hexToRgb(col)
```

### Arguments

| | |
|---|---|
| col | character vector of hexadecimal colors. |

### Value

Integer matrix with RGB rows.

### See Also

color-conversion-overview

### Examples

```
hexToRgb(c("#A52A2A", "#FFDAB9"))

# an alpha channel is kept
hexToRgb("#A52A2A80")
```

## htmlNotation *HTML notation for hat and bar diacritics*

### Description

Append the HTML entity for a circumflex ("hat", as in an estimator $\hat{x}$) or macron ("bar", as in a sample mean $\bar{x}$) to a label.

### Usage

```
htmlHat(x)

htmlBar(x)
```

### Arguments

| | |
|---|---|
| x | a character vector, typically a variable name or symbol |

### Value

a character vector with the diacritic's HTML entity appended

### See Also

Other html: asFileLink(), asHtml(), asImg(), embedFile(), escapeHtml(), htmlSubscript, preview(), preview.html(), toHtmlTable()

### Examples

```
htmlHat("p")  # -> "p&#770;"   (renders as p with a circumflex)
htmlBar("x")  # -> "x&#772;"   (renders as x with a macron)
```

---

## htmlSubscript *Subscript notation*

---

### Description

Infix operator producing an HTML subscript, e.g. for indexed symbols such as $x_i$.

### Usage

```
x %_% i
```

### Arguments

| | |
|---|---|
| x | a character vector (the base symbol) |
| i | a character vector (the subscript), recycled against x |

### Value

a character vector: x followed by <sub>i</sub>

### See Also

Other html: asFileLink(), asHtml(), asImg(), embedFile(), escapeHtml(), htmlNotation, preview(), preview.html(), toHtmlTable()

### Examples

```
"x" %_% "i"  # -> "x<sub>i</sub>"
```

`isValidPlotRegion` *Check Whether the Current Plot Region Is Large Enough*

## Description

Tests whether the plot region of the active graphics device meets a minimal size requirement. This is useful before drawing into small devices or heavily subdivided layouts (`mfrow`, `layout()`, Shiny render panes), where an undersized region would otherwise fail with the rather cryptic error `"figure margins too large"`.

## Usage

```
isValidPlotRegion(minPin = .getOption("plot.minPin", c(0.5, 0.5)))
```

## Arguments

| | |
|---|---|
| `minPin` | numeric vector of length 2, giving the minimal required width and height of the plot region in inches. Defaults to the option `DescToolsX.plot.minPin` and falls back to `c(0.5, 0.5)` if the option is not set. |

## Details

The plot region is the area inside the figure margins, as reported by `par("pin")`. Its size results from the device dimensions minus the current margins (`mar`, `oma`), so both a small device and oversized margins can render it degenerate.

If no graphics device is open (`dev.cur() == 1`), the function returns `TRUE`, since the next high-level plot call will open a fresh device with default dimensions. Note that querying `par()` in this state would itself open a device as a side effect, which this function deliberately avoids.

The function has no side effects and never throws; it is meant as a guard for conditional plotting, e.g. `if (isValidPlotRegion()) plot(x) else message("device too small")`.

## Value

a single logical: `TRUE` if the plot region is at least `minPin` inches in both dimensions (or if no device is open), `FALSE` otherwise.

## See Also

`par()` (entries `pin`, `fin`, `mar`), `dev.cur()`

Other graphics.layout: `abcCoords()`, `axTicksPosixct()`, `axisBreak()`, `lineToUser()`, `mar()`, `plotFacet()`, `spreadOut()`

### Examples

```
# guard a plot against a degenerate region
if (isValidPlotRegion()) {
  plot(rnorm(20))
} else {
  message("plot region too small")
}

# require a larger region, e.g. for a wide correlation plot
isValidPlotRegion(minPin = c(4, 3))
```

---

## lighten *Lighten Colors*

---

### Description

Lighten colors by mixing them with white.

### Usage

```
lighten(col, amount = 0.2)
```

### Arguments

| | |
|---|---|
| col | vector of valid R colors. |
| amount | numeric value between 0 and 1 specifying the amount of lightening. A value of 0 leaves the color unchanged, while 1 returns white. |

### Details

Colors are mixed linearly with white in RGB space:

$$x_{new} = x + amount \cdot (255 - x)$$

### Value

Character vector of hexadecimal colors.

### See Also

Other color.manipulation: addOpacity(), colToOpaque(), darken(), fade(), mixCol()

## Examples

```
lighten("tomato")
lighten(c("tomato", "royalblue"), amount = 0.5)

# increasing amounts
amount <- seq(0, 1, 0.2)
barplot(rep(1, 6), col = sapply(amount, lighten, col = "tomato"),
        names.arg = amount, yaxt = "n", xlab = "amount")
```

---

# lines.lm *Add a Linear Regression Line*

---

## Description

Add a linear regression line to an existing plot. The function first calculates predictions from an `lm` object and then adds the fitted line together with optional confidence and prediction bands.

## Usage

```
## S3 method for class 'lm'
lines(
  x,
  col = pal()[1],
  lwd = 2,
  lty = "solid",
  type = "l",
  n = 100,
  cband = list(conf.level = 0.95),
  pband = NA,
  xpred = NULL,
  ...
)

## S3 method for class 'lmlog'
lines(
  x,
  col = pal()[1],
  lwd = 2,
  lty = "solid",
  type = "l",
  n = 100,
  cband = list(conf.level = 0.95),
  pband = NA,
  xpred = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | linear model object as returned by lm(). |
| col | line color. Defaults to pal()[1]. |
| lwd | line width. |
| lty | line type. |
| type | plotting type passed to lines(). |
| n | number of points used for plotting the fit. |
| cband | controls the confidence band. May be TRUE, FALSE, NULL, NA, or a named list. The confidence level is specified via conf.level, all other elements are graphical parameters of the band. Default is list(conf.level=0.95). |
| pband | controls the prediction band, as cband. Default is NA. |
| xpred | optional numeric vector defining the range over which predictions should be calculated. |
| ... | further graphical parameters passed to lines() for the fitted line. |

## Details

In contrast to abline(), polynomial models and transformed predictors are supported as long as the model contains exactly one predictor variable. A polynomial, y ~ poly(x, 2) or y ~ x + I(x^2), is drawn against x; a single transformed term such as y ~ log(x) is drawn against the transformed values, matching plot(y ~ log(x)).

Confidence and prediction bands are controlled via cband and pband. These arguments can be:

- FALSE, NULL or NA: suppress the band
- TRUE: draw the band with default settings
- a named list: customize the band appearance and confidence level

## Value

No return value; called for its side effect.

## See Also

lines(), lm()

Other graphics.trendlines: lines.loess(), splineCI

## Examples

```
plot(dist ~ speed, cars)
lines(lm(dist ~ speed, cars))

# polynomial model with a 99% confidence band and a prediction band
plot(dist ~ speed, cars)
lines(lm(dist ~ poly(speed, 2), cars), col = "red",
      cband = list(conf.level = 0.99), pband = TRUE)
```

```
# transformed predictor, no bands
plot(mpg ~ log(hp), mtcars)
lines(lm(mpg ~ log(hp), mtcars), cband = FALSE)
```

---

`lines.loess` *Add a Loess Smoother and Its Confidence Band*

---

## Description

Add a loess smoother to an existing plot. The function first calculates predictions from a `loess` object and then adds the fitted smoother together with an optional confidence band.

## Usage

```
## S3 method for class 'loess'
lines(
  x,
  col = .useTheme,
  lwd = 2,
  lty = "solid",
  type = "l",
  n = 100,
  band = list(conf.level = 0.95),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | a fitted `loess()` object. |
| `col` | line color of the smoother. `.useTheme` (default) resolves to `getTheme()$twin[1]` - the first of the theme's two-color pair (see theme). |
| `lwd` | line width. |
| `lty` | line type. |
| `type` | plotting type passed to `lines()`. |
| `n` | number of points used for plotting the fit. |
| `band` | controls the confidence band. May be `TRUE`, `FALSE`, `NULL`, `NA`, or a named list. The confidence level is specified via `conf.level`, all other elements are graphical parameters of the band. Default is `list(conf.level = 0.95)`. |
| `...` | further graphical parameters passed to `lines()` for the smoother. |

## Details

The confidence band is controlled via band. This argument may be:

- `FALSE`, `NULL` or `NA`: suppress the band
- `TRUE`: draw the band with default settings
- a named list: customize the band appearance and confidence level

## Value

No return value (invisible NULL), called for its side effect of adding the smoother and its confidence band to the current plot.

## Note

Loess can result in substantial computational load for large datasets.

## See Also

loess(), scatter.smooth(), smooth.spline()

Other graphics.trendlines: lines.lm(), splineCI

## Examples

```
x <- runif(100)
y <- rnorm(100)

plot(x, y)
lines(loess(y ~ x))

plot(dist ~ speed, cars)
lines(
  loess(dist ~ speed, cars),
  band = list(
    conf.level = 0.99,
    col = addOpacity("red", 0.4),
    border = "black"
  )
)
```

---

# lineSep *Create a Line Separator String*

---

## Description

Generates a character string that can be used as a horizontal line separator in console output.

## Usage

```
lineSep(sep = .getOption("linesep"))
```

## Arguments

| | |
|---|---|
| sep | A character string used as separator. If NULL, a default Unicode box-drawing character ("\u2500") is used. If sep consists of a single visible character, it is repeated to match the current console width (minus two characters). |

### Details

ANSI escape sequences (e.g., color codes) are removed using `cli::ansi_strip()` when determining the visible width.

The console width is determined via `getOption("width", 80)`.

### Value

A character string representing a horizontal separator line.

### Examples

```
lineSep()
lineSep("-")
lineSep(cli::col_red("="))
```

---

## lineToUser *Convert Line Coordinates To User Coordinates*

---

### Description

Functions like `mtext` or `axis` use the `line` argument to set the distance from plot. Sometimes it's useful to have the distance in user coordinates. `lineToUser()` does this nontrivial conversion.

### Usage

```
lineToUser(line, side)
```

### Arguments

| | |
|---|---|
| `line` | the number of lines |
| `side` | the side of the plot |

### Details

For the `lineToUser` function to work, there must be an open plot.

### Value

the user coordinates for the given lines

### See Also

`mtext()`

Other graphics.layout: `abcCoords()`, `axTicksPosixct()`, `axisBreak()`, `isValidPlotRegion()`, `mar()`, `plotFacet()`, `spreadOut()`

## Examples

```
plot(1:10)
lineToUser(line=2, side=4)
```

---

longToRgb *Convert Long Integers to RGB*

---

## Description

Decode long integers into RGB values.

## Usage

```
longToRgb(col)
```

## Arguments

| | |
|---|---|
| col | integer vector. |

## Value

RGB matrix.

## See Also

color-conversion-overview

## Examples

```
longToRgb(c(255, 65280, 16711680))

# there and back again
rgbToHex(longToRgb(rgbToLong(colToRgb("tomato"))))
```

mar *Get or set plot margins conveniently*

## Description

Convenience wrapper around graphics::par() for getting and setting plot margins (mar) or outer margins (oma). Individual sides can be modified without affecting the others.

## Usage

```
mar(bottom = NULL, left = NULL, top = NULL, right = NULL, outer = FALSE)
```

## Arguments

| | |
|---|---|
| bottom, left, top, right | numeric scalars specifying the margin size (in lines) for each side. If NULL, the current value is retained. |
| outer | logical; if TRUE, outer margins (oma) are used instead of inner margins (mar). |

## Details

This function simplifies partial updates of par("mar") or par("oma"). It avoids the need to manually query and reconstruct the full margin vector.

For restoring graphical parameters, the recommended base R approach is:

```
op <- par(no.readonly = TRUE)
on.exit(par(op))
```

## Value

If no arguments are provided, returns the current margin vector (numeric of length 4). Otherwise, sets the margins and returns the new values invisibly.

## See Also

Other graphics.layout: abcCoords(), axTicksPosixct(), axisBreak(), isValidPlotRegion(), lineToUser(), plotFacet(), spreadOut()

## Examples

```
# Get current margins
mar()

# Set bottom margin only
mar(bottom = 2)

# Set multiple margins
mar(bottom = 2, left = 2)
```

```
# Modify outer margins
mar(top = 1, outer = TRUE)
```

---

`mixCol` *Mix Colors*

---

## Description

Mix two sets of colors in RGB space.

## Usage

```
mixCol(col1, col2, weights = 0.5)
```

## Arguments

| | |
|---|---|
| `col1` | first vector of colors. |
| `col2` | second vector of colors. |
| `weights` | numeric value between 0 and 1 specifying the contribution of `col2`. A value of 0 returns `col1`, while 1 returns `col2`. |

## Details

Colors are mixed linearly in RGB space:

$$x_{mix} = (1 - weights)x_1 + weights x_2$$

All arguments are recycled as necessary.

## Value

Character vector of hexadecimal colors.

## See Also

Other color.manipulation: `addOpacity()`, `colToOpaque()`, `darken()`, `fade()`, `lighten()`

## Examples

```
mixCol("red", "blue")
mixCol("red", "blue", weights = 0.2)

# the shorter vector is recycled
mixCol(c("red", "yellow"), "blue")

# from red to blue
w <- seq(0, 1, 0.1)
```

```
barplot(rep(1, 11), col = sapply(w, mixCol, col1 = "red", col2 = "blue"),
        names.arg = w, yaxt = "n", xlab = "weights")
```

---

`pal` *Get a Color Palette*

---

## Description

Returns `n` colors from a named palette. All palettes always return exactly `n` colors regardless of their base size:

- `n < length(base)`: evenly spaced sample for maximum contrast
- `n = length(base)`: returned as-is
- `n > length(base)`: interpolated via `grDevices::colorRampPalette()`

## Usage

```
pal(name, n = NA, opacity = 1)

## S3 method for class 'Palette'
plot(x, main = NULL, cex = 0.85, border = "grey70", stamp = .useTheme, ...)
```

## Arguments

| | |
|---|---|
| `name` | character or integer. Palette name (full match via `base::match.arg()`) or index into `palNames()`. If missing, returns the palette named in the active theme (`getTheme()$palette`, see theme). |
| `n` | integer, number of colors to return. Default `NA` returns the colors as contained in the palette. |
| `opacity` | numeric in $[0, 1]$, opacity. Default 1 (opaque). Applied via `grDevices::adjustcolor()`. |
| `x` | palette object to be plotted. |
| `main` | main title. `NULL` (default) uses the name of the palette, regardless of the theme's `autoTitle`; `""`, `NA` or `FALSE` suppress the title and compact the top margin. |
| `cex` | character expansion of the labels (index, name, hex code). It is reduced automatically when the rows get too narrow, and the labels are left out for very long palettes. |
| `border` | color for the border of the swatches. |
| `stamp` | controls the corner stamp. `.useTheme` (default) resolves to `getTheme()$stamp`. `TRUE`/`FALSE`/`NULL`, a string, or a named list for `stamp()`. |
| `...` | further graphical parameters, passed to `par()` via the internal framework. |

## Value

a character vector of `n` hex color codes of class `c("palette", "character")` with a `"name"` attribute.

## See Also

palNames(), grDevices::colorRampPalette(), grDevices::adjustcolor()

Other color.palettes: hcol(), palNames()

## Examples

```
# default palette from options
pal()

# discrete palette — 3 maximally contrasting colors
pal("dark", n = 3)

# continuous gradient — 50 colors
pal("red-white-blue-2", n = 50)

# by index
pal(1, n = 10)

# with transparency
pal("helsana", n = 5, opacity = 0.5)

# show all names
palNames()
```

---

# palNames *List Available Palette Names*

---

## Description

Returns the names of all palettes available in pal(), optionally filtered by type.

## Usage

```
palNames(type = c("all", "continuous", "discrete"))
```

## Arguments

| | |
|---|---|
| type | character, one of "all" (default), "continuous", or "discrete". |

## Value

a character vector of palette names.

## See Also

pal()

Other color.palettes: hcol(), pal()

## Examples

```
palNames()
palNames("continuous")
palNames("discrete")
```

---

`plot.BlandAltman` *Bland-Altman Plot*

---

## Description

Plots a Bland-Altman agreement analysis.

## Usage

```
## S3 method for class 'BlandAltman'
plot(
  x,
  main = "",
  xlab = "Mean",
  ylab = "Difference",
  xlim = NULL,
  ylim = NULL,
  pch = 16,
  col = NULL,
  grid = TRUE,
  meanLine = TRUE,
  limits = TRUE,
  cband = FALSE,
  trend = FALSE,
  showText = TRUE,
  stamp = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `"BlandAltman"`. |
| `main` | plot title. |
| `xlab` | X-axis label. |
| `ylab` | Y-axis label. |
| `xlim` | X-axis limits. |
| `ylim` | Y-axis limits. |
| `pch` | plotting symbol. |
| `col` | point colour. |

| | |
|---|---|
| grid | logical; draw a background grid. |
| meanLine | logical; draw the bias line. |
| limits | logical; draw the limits of agreement. |
| cband | logical; draw confidence bands. |
| trend | logical; draw a regression line of differences versus means. |
| showText | logical; annotate bias and limits. |
| stamp | optional plot stamp. |
| ... | further arguments passed to plot(). |

## Details

Objects of class "BlandAltman" are typically created with DescToolsX::blandAltmanData(). The method uses the elements mean and diff (one value per pair), bias, loaLower and loaUpper, and for the confidence bands biasCI, loaLowerCI and loaUpperCI (each a vector of length 2).

## Value

Invisibly returns x.

## See Also

Other plot.s3: plot.Desc.qn(), plot.Desc.table(), plot.Lc()

## Examples

```
# two measurement methods applied to the same 40 subjects; the object is
# built by hand here, normally DescToolsX::blandAltmanData() returns it
set.seed(1)
a <- rnorm(40, mean = 100, sd = 15)
b <- a + rnorm(40, mean = 2, sd = 5)

d   <- a - b
n   <- length(d)
s   <- sd(d)
tq  <- qt(0.975, df = n - 1)
loa <- mean(d) + c(-1, 1) * 1.96 * s
seLoa <- sqrt(3 * s^2 / n)

ba <- structure(list(
  mean       = (a + b) / 2,
  diff       = d,
  bias       = mean(d),
  biasCI     = mean(d) + c(-1, 1) * tq * s / sqrt(n),
  loaLower   = loa[1],
  loaUpper   = loa[2],
  loaLowerCI = loa[1] + c(-1, 1) * tq * seLoa,
  loaUpperCI = loa[2] + c(-1, 1) * tq * seLoa
), class = "BlandAltman")
```

```
plot(ba, main = "Method A vs. method B")
plot(ba, cband = TRUE, trend = TRUE)
```

---

`plot.Desc.qn` *Plot Method for Numeric-Categorical* `Desc` *Objects*

---

## Description

Visualises the relationship between a numeric variable and a categorical variable, as computed by `DescToolsX::desc(y ~ x)` (or `x ~ y`) for a numeric/categorical pair. Five panel types are available, selectable (and combinable) via `which`.

## Usage

```
## S3 method for class 'Desc.qn'
plot(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  which = 2,
  verbose = NULL,
  col = .useTheme,
  box = .useTheme,
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `"Desc.qn"`, as returned by `DescToolsX::desc()` for a numeric-categorical pair. |
| `main` | main title. `NULL` (default) derives a title per panel from the variable names and the panel type (e.g. `"area ~ delivery_min (Spineplot)"`), regardless of the theme's `autoTitle`. `""`, `NA`, or `FALSE` suppress the title and compact the top margin. Any other string is used as-is, identically for every selected panel. |
| `xlab` | x-axis label. `NULL` (default) derives a label specific to each panel: the numeric variable for panels 1, 2, 3 and 5, the categorical variable for panel 4 (boxplot, groups on the x-axis). Supplying a value overrides this for every selected panel. |
| `ylab` | y-axis label. `NULL` (default) derives a label specific to each panel (e.g. `"P(y)"` for the conditional density, `"Density"` for the grouped density plot). Supplying a value overrides this for every selected panel. |
| `which` | integer vector selecting one or more panels to draw, in the given order. One or more of: |

1 Conditional density plot (graphics::cdplot()).

2 Spineplot (graphics::spineplot()). Default.

3 Overlapping kernel density curves, one per group (via plotDens()).

4 Boxplot of the numeric variable by group (via plotBox()).

5 Prevalence (with Wilson confidence intervals) across quantile bins of the numeric variable. Only available when the categorical variable is binary (k == 2); a message is issued and the panel skipped otherwise.

Selecting multiple panels does not change the plotting layout (no implicit mfrow) - arranging multiple panels on one device is left to the caller (e.g. par(mfrow = c(2, 1)) beforehand).

verbose — integer; currently computed from x$meta$verbose/getOption("DescTools.verbose") for consistency with other plot.Desc.* methods, but not yet consulted to pick a default which.

col — color specification. .useTheme (default) resolves a panel-appropriate default rather than one shared color, since fill ramps, categorical sets, and single accents are different things:

**panels 1/2** a grey ramp from "grey30" to "grey90", sized to the number of categorical levels (not theme-driven by design, to keep the unordered category fill neutral - see plotMosaic() for the same rationale).

**panels 3/4** pal(getTheme()$palette), the active theme's qualitative palette, sized to the number of levels.

**panel 5** getTheme()$twin[1], the active theme's primary accent color.

Supplying col explicitly overrides the default uniformly for every selected panel.

box — controls the plot frame for panels 4 and 5. .useTheme (default) follows the active theme (getTheme()$box); FALSE/NA suppress it; a named list overrides graphics::box() arguments. Panels 1/2 (cdplot()/spineplot()) have no effect from this argument - they always draw their native frame unconditionally, with no toggle to override it. Panel 4 (plotBox()) always draws its own frame regardless of this argument. Panel 3 (plotDens()) never draws a frame, regardless of this argument.

legend — controls the legend for panel 3 (grouped density curves). TRUE (default) draws a legend with the group levels. FALSE/NA suppresses it. A named list overrides arguments forwarded to graphics::legend(). Has no effect on the other panels.

stamp — controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp, drawn once after all selected panels (panels 3/4 delegate to plotDens()/plotBox(), whose own stamp is suppressed internally to avoid a duplicate). TRUE/FALSE/NULL, a string, or a named list for stamp().

... — further graphical parameters, passed to par() via the internal framework and to the underlying panel-drawing functions (cdplot(), spineplot(), plotDens(), plotBox(), or graphics::plot(), depending on the selected panel).

## Details

The left margin is sized automatically from the longest of: the (possibly panel-specific) y-axis labels, and - for panels 1/2 - the categorical level names drawn as axis tick labels, so neither is ever clipped regardless of which.

## Value

Invisibly returns `x`.

## See Also

`DescToolsX::desc`, `plotDens()`, `plotBox()`, `graphics::cdplot()`, `graphics::spineplot()`

Other plot.s3: `plot.BlandAltman()`, `plot.Desc.table()`, `plot.Lc()`

## Examples

```
# the object is built by hand here; normally DescToolsX::desc(y ~ x)
# returns it for a numeric-categorical pair
d <- structure(list(
  res  = list(xOk  = ToothGrowth$len,
              yOk  = ToothGrowth$supp,
              lvls = levels(ToothGrowth$supp),
              k    = nlevels(ToothGrowth$supp)),
  meta = list(xname = "len", yname = "supp")
), class = "Desc.qn")

plot(d)                 # spineplot
plot(d, which = 3)      # density by group
plot(d, which = 4)      # boxplot by group
```

---

`plot.Desc.table` *Plot Method for Categorical-Categorical* `Desc` *Objects*

---

## Description

Visualises a (two-dimensional) cross-tabulation, as computed by `DescToolsX::desc()` for a categorical/categorical pair. Four panel types are available, selectable (and combinable) via `which`. Higher-dimensional tables (more than two margins) are not supported; a message is issued and the call returns invisibly.

## Usage

```
## S3 method for class 'Desc.table'
plot(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  which = 1,
  verbose = NULL,
  col = .useTheme,
  box = .useTheme,
  stamp = .useTheme,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| x | an object of class "Desc.table", as returned by DescToolsX::desc() for a categorical-categorical pair. |
| main | main title. NULL (default) derives a title per panel from x$meta$xname (the deparsed expression originally passed to desc(), e.g. "table(Pizza$area, Pizza$driver)") combined with a panel-type label for context when multiple panels are shown (e.g. "table(Pizza$area, Pizza$driver) (Spineplot)"). There is no y ~ x pair to draw on here - x$meta carries only a single xname, since a table built outside a two-sided formula has no separately named "x" and "y" variable. Drawn regardless of the theme's autoTitle. "", NA, or FALSE suppress the title and compact the top margin. Any other string is used as-is, identically for every selected panel. |
| xlab | x-axis label. NULL (default) leaves the panel's own default in place (empty for the spineplot, the table name for the mosaic plots, the column dimnames for the association plot). Supplying a value overrides this for every selected panel. |
| ylab | y-axis label. NULL (default) leaves the panel's own default in place (typically empty/unlabeled, since the row dimension of a table built via e.g. table(a, b) usually has no name carried in x$meta). Supplying a value overrides this for every selected panel. |
| which | integer vector selecting one or more panels to draw, in the given order. One or more of:<br>1 Spineplot (graphics::spineplot()). Default.<br>2 Mosaic plot (via plotMosaic()).<br>3 Mosaic plot (swapped axis).<br>4 Association plot (Cohen-Friendly plot) via plotAssoc().<br>5 Heatmap of cell proportions (via plotHeatmap(), scale = "prop").<br>Selecting multiple panels does not change the plotting layout (no implicit mfrow) - as with other plot.Desc.* methods, arranging multiple panels on one device is left to the caller (e.g. par(mfrow = c(2, 1)) beforehand). |
| verbose | integer; currently computed from x$meta$verbose/getOption("DescTools.verbose") for consistency with other plot.Desc.* methods, but not yet consulted to pick a default which. |
| col | color specification. .useTheme (default) resolves a panel-appropriate default rather than one shared color, since fill ramps, diverging palettes, and sequential heat scales are different things:<br>**panel 1** a grey ramp from "grey30" to "grey90", sized to the number of rows of tab - the panel draws spineplot(t(tab)), so the stacked (filled) dimension is the row dimension of tab, not its columns (not theme-driven by design, to keep the unordered category fill neutral).<br>**panel 2** a grey ramp from "grey30" to "grey90", sized to the number of columns of tab (the fill dimension of the untransposed mosaic), passed to plotMosaic(). |

**panel 3** a grey ramp from `"grey30"` to `"grey90"`, sized to the number of rows of `tab` - with `swap = TRUE` the fill dimension is the row dimension, passed to `plotMosaic()`.

**panel 4** left at `plotAssoc()`'s own default (`pal("red-white-blue-3", n = 100)`), a diverging palette - cell colors there encode the sign and strength of Pearson residuals, so a categorical or grey-ramp default would not be meaningful. Supplying `col` overrides this with the diverging palette of the user's choice.

**panel 5** left at `plotHeatmap()`'s own default (`pal("Blues", n = 100)`), a sequential ramp - cell colors there encode magnitude only. Supplying `col` overrides this.

Supplying `col` explicitly overrides the default uniformly for every selected panel.

`box` controls the plot frame. `.useTheme` (default) follows the active theme (`getTheme()$box`); `FALSE`/`NA` suppress it; a named list overrides frame-drawing arguments.

**panel 1** has no effect - `spineplot()` always draws its native frame unconditionally, with no toggle to override it.

**panels 2/3** `plotMosaic()` always draws its own frame; this argument has no effect.

**panel 4** `plotAssoc()` has no frame/box concept of its own (it draws dashed reference lines instead); this argument has no effect.

**panel 5** forwarded as-is to `plotHeatmap()`'s own `box` argument, which draws the outer frame via `rect()` at the exact tile boundaries rather than `graphics::box()`.

`stamp` controls the corner stamp. `.useTheme` (default) resolves to `getTheme()$stamp`, drawn once after all selected panels. Panels 2-5 delegate to `plotMosaic()`/`plotAssoc()`/`plotHeatmap()`, whose own `stamp` argument is set to `NA` internally to avoid a duplicate. `TRUE`/`FALSE`/`NULL`, a string, or a named list for `stamp()`.

`...` further graphical parameters, passed to `par()` via the internal framework and to the underlying panel-drawing functions (`spineplot()`, `plotMosaic()`, `plotAssoc()`, or `plotHeatmap()`, depending on the selected panel).

## Details

The left margin is sized automatically from the longest of: the y-axis label, and - for panels 1/2 - the row names of `tab` drawn as axis tick labels, so neither is ever clipped regardless of `which`.

Only two-dimensional tables are supported. If `x` carries a table with more than two margins, a message is issued and the function returns invisibly without drawing anything.

## Value

Invisibly returns `x`.

## See Also

`DescToolsX::desc`, `plotAssoc()`, `plotHeatmap()`, `plotMosaic()`, `graphics::spineplot()`

Other plot.s3: `plot.BlandAltman()`, `plot.Desc.qn()`, `plot.Lc()`

## Examples

```
# the object is built by hand here; normally DescToolsX::desc()
# returns it for a pair of categorical variables
d <- structure(list(
  tab  = margin.table(HairEyeColor, c(1, 2)),
  meta = list(xname = "Hair ~ Eye")
), class = "Desc.table")

plot(d)                 # spineplot
plot(d, which = 2)      # mosaic plot
```

---

# plot.Lc *Plot Methods for Lorenz Curve Objects*

---

## Description

Visualize objects of class "Lc" and "LcList" returned by DescToolsX::lc(). The plot() method draws a new Lorenz curve plot including the line of perfect equality; lines() and points() add to an existing plot.

## Usage

```
## S3 method for class 'Lc'
plot(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  general = FALSE,
  col = NULL,
  line = TRUE,
  points = NULL,
  eqline = TRUE,
  grid = .useTheme,
  box = .useTheme,
  cband = NA,
  stamp = .useTheme,
  ...
)

## S3 method for class 'Lc'
lines(x, general = FALSE, col = NULL, lwd = 2, lty = 1, cband = NA, ...)

## S3 method for class 'Lc'
```

```
points(x, general = FALSE, pch = 16, col = NULL, ...)

## S3 method for class 'LcList'
lines(x, col = NULL, ...)

## S3 method for class 'LcList'
points(x, col = NULL, ...)

## S3 method for class 'LcList'
plot(x, col = NULL, general = FALSE, ylim = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | object of class "Lc" (for plot.Lc(), lines.Lc(), points.Lc()) or "LcList" (for the *.LcList() methods). |
| main, xlab, ylab | main title and axis labels, used by plot.Lc() only. All default to NULL: no title, "p", and "L(p)" ("GL(p)" if general = TRUE), respectively. |
| xlim, ylim | numeric vectors of length 2 giving axis limits, used by plot.Lc() and plot.LcList(). Default NULL, which resolves to c(0, 1) for xlim and, for ylim, to c(0, 1) for the standard and c(0, max(L)) for the generalized curve. |
| general | logical. If TRUE, the generalized Lorenz curve (scaled by the mean) is displayed instead of the standard curve. Default is FALSE. |
| col | color of curve and symbols. For plot.Lc(), lines.Lc() and points.Lc() a single color (default NULL, i.e. "black" in plot.Lc() and the device default in the low-level methods). For the "LcList" methods a vector recycled to the number of groups (default NULL, i.e. seq_len(k)). |
| line | logical or list, used by plot.Lc() to control drawing of the Lorenz curve. TRUE (default) draws it with package defaults (lty = 1, lwd = 2); FALSE suppresses it (and the confidence band); a list overrides individual defaults and is forwarded to lines.Lc(). |
| points | NULL, logical or list, used by plot.Lc() to control drawing of symbols on the curve. NULL (default) is automatic: symbols are drawn only while the curve has at most getOption("DescToolsX.plot.maxSymbols") knots (100 by default), which keeps large samples legible. TRUE always draws them with package defaults (pch = 21, bg = "white", cex = 1.4); FALSE suppresses them; a list overrides individual defaults and is forwarded to points.Lc(). |
| eqline | logical or list, used by plot.Lc() only. Controls the line of perfect equality: TRUE (default) draws it with package defaults (col = "grey50", lty = 2), FALSE suppresses it, a list is forwarded to graphics::abline(). Its slope is 1 for the standard and max(L) for the generalized curve; overriding a/b is possible but rarely sensible. |
| grid, box | callIf-style specs for the grid and the box around the plot region, used by plot.Lc() only. .useTheme (default) lets getTheme() decide, TRUE/FALSE force drawing/suppression, and a named list is forwarded to graphics::grid() resp. graphics::box(). |
| cband | used by plot.Lc() and lines.Lc(). NA to suppress the confidence band (default), or a list of arguments passed to DescToolsX::predict.Lc() to control bootstrap confidence intervals. |

| | |
|---|---|
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further arguments. For plot.Lc(), graphical parameters passed to graphics::par() via .applyParFromDots() (e.g. mar, cex.axis, las). For lines.Lc() and points.Lc(), further arguments passed on to graphics::lines() and graphics::points(), respectively. For plot.LcList(), arguments are passed to plot.Lc() for the first group and, restricted to those the low-level method understands, to lines.Lc() for the remaining ones. |
| lwd | line width, used by plot.Lc() (via line) and lines.Lc(). Default is 2. |
| lty | line type, used by plot.Lc() (via line) and lines.Lc(). Default is 1. |
| pch | plotting symbol, used by points.Lc() only. Default is 16. |

## Details

For "LcList" objects (grouped Lorenz curves), plot() draws the first group and overlays the remaining groups with lines(). Colors cycle automatically when col is not supplied and are recycled to the number of groups otherwise.

The curve of plot.Lc() is drawn by lines.Lc() and the symbols by points.Lc(), so all three methods share one code path and one set of semantics - including the confidence band, which is controlled by cband in plot.Lc() exactly as it is in lines.Lc(). Pass a list of arguments to DescToolsX::predict.Lc() to control the bootstrap (e.g. cband = list(conf.level = 0.90, n = 500)). Set cband = NA (default) to suppress the band. Note that line = FALSE suppresses the band along with the curve.

With general = TRUE the generalized Lorenz curve is displayed. It ends at the mean rather than at 1, so the default ylim and the slope of the equality line follow the data; for "LcList" objects the panel is sized to accommodate *all* groups, not just the first.

## Value

All methods return NULL invisibly.

## See Also

DescToolsX::lc() for computing the Lorenz curve, DescToolsX::predict.Lc() for bootstrap confidence intervals, DescToolsX::gini() for the Gini coefficient.

Other plot.s3: plot.BlandAltman(), plot.Desc.qn(), plot.Desc.table()

## Examples

```
# Lorenz curve objects are built by hand here; normally
# DescToolsX::lc() returns them
lorenz <- function(v) {
  v <- sort(v)
  L <- c(0, cumsum(v)) / sum(v)
  structure(list(p = c(0, seq_along(v)) / length(v),
                 L = L, L.general = L * mean(v)),
            class = "Lc")
}
```

```
set.seed(4)
even   <- lorenz(rgamma(50, shape = 8))
skewed <- lorenz(rlnorm(50, sdlog = 1.2))

plot(skewed, main = "Lorenz curves")
lines(even, col = "steelblue")

# several groups at once, generalized curves
grp <- structure(list(even = even, skewed = skewed), class = "LcList")
plot(grp, general = TRUE, col = c("steelblue", "darkred"))
```

---

plotArea *Stacked Area Plot*

---

## Description

Draws one or several stacked area series using cumulative polygons. The function accepts either a matrix of values or separate x and y coordinates. Multiple series are displayed as stacked areas.

## Usage

```
plotArea(
  x,
  y,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  prop = FALSE,
  col = NULL,
  grid = .useTheme,
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector, matrix or data frame. If y is missing, x is interpreted as a matrix of series where rows correspond to x positions and columns to individual areas. |
| y | optional numeric vector or matrix giving the y-values. If supplied, x is interpreted as the x-coordinates. |
| main | main title of the plot. |
| xlab | label for the x-axis. |

| | |
|---|---|
| ylab | label for the y-axis. |
| xlim | limits for the x-axis. |
| ylim | limits for the y-axis. |
| prop | logical indicating whether rows should be converted to proportions so that stacked areas sum to one. |
| col | fill colours used for the areas. |
| grid | controls the background grid. .useTheme (default) follows the active theme (getTheme()$grid), TRUE draws it, FALSE, NULL or NA suppress it, and a named list is passed to graphics::grid(). |
| legend | logical or list controlling the legend. If TRUE, a legend is drawn using the column names of the data. If a list is supplied, its elements are passed to the internal legend drawing routine. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to graphics::par() via .applyParFromDots() and to the plotting functions. |

## Details

If y is missing, x is interpreted as a matrix and each column is drawn as a separate stacked area.

The cumulative sums are calculated row-wise and displayed as polygons stacked on top of each other.

If prop = TRUE, each row is converted to proportions before plotting, so the stacked areas sum to one.

Row names are used as x-axis labels when available and y is omitted.

## Value

Invisibly returns a list containing:

- x the x-values used for plotting,
- y the original y-values,
- cumulative the cumulative values used to construct the areas,
- legend the legend specification if drawn.

## See Also

Other plot.univariate: plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
plotArea(VADeaths)

plotArea(
  WorldPhones,
  col = pal("helsana")
)

plotArea(
  WorldPhones,
  prop = TRUE,
  col = rainbow(ncol(WorldPhones))
)

x <- 1:20
y <- cbind(
  A = runif(20, 1, 5),
  B = runif(20, 1, 3),
  C = runif(20, 1, 4)
)

plotArea(x, y)
```

---

# plotAssoc *Association Plot for Contingency Tables*

---

## Description

Plots an association plot (Cohen-Friendly plot) for a two-dimensional contingency table. Cells are drawn with widths proportional to the square root of expected frequencies and heights proportional to Pearson residuals. Color encodes both the direction and strength of the association using a diverging palette.

## Usage

```
plotAssoc(
  x,
  main = NULL,
  xlab = TRUE,
  ylab = TRUE,
  space = 0.3,
  reorder = TRUE,
  col = pal("red-white-blue-3", n = 100L),
  border = NA,
  labels = FALSE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a two-dimensional contingency table (table or matrix). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the expression passed as x (e.g. "tab"). "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | character, x-axis label. Defaults to the first dimension name. |
| ylab | character, y-axis label. Defaults to the second dimension name. |
| space | numeric, fraction of average cell width/height used as gap between cells. Default 0.3. |
| reorder | logical. If TRUE (default), rows and columns are reordered by the strength of the strongest association (max(\|residual\|)) in descending order. |
| col | character vector of colors for the diverging palette. Default uses pal("red-white-blue-3", n = 100) from the DescToolsX theme. Negative residuals map to the first color, zero to the middle, positive to the last. |
| border | the color of the border |
| labels | logical or character. If TRUE, Pearson residuals are printed inside each cell. If FALSE (default), no labels are shown. A character format string (e.g. "\%.1f") can also be passed for custom formatting. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, or an explicit string, as for .withGraphicsState() (internal). |
| ... | further arguments passed to graphics::rect(). |

## Details

The plot is based on the association plot described in Cohen (1980) and Friendly (1992). Each cell $(i, j)$ is represented by a rectangle:

- **width** proportional to $\sqrt{e_{ij}}$ (square root of expected frequency)
- **height** proportional to the Pearson residual $d_{ij} = (f_{ij} - e_{ij})/\sqrt{e_{ij}}$

A horizontal baseline at zero represents independence. Cells above the baseline indicate positive association, cells below negative association.

Color encodes both direction and magnitude: the diverging palette runs from the negative color (strong negative residual) through white (no association) to the positive color (strong positive residual).

## Value

A list with the components x and y, returned invisibly: the horizontal centres of the bars, one per column of the table, and the baselines of the residuals, one per row. Both are named by the levels and given in the order in which they are drawn.

## References

Cohen, A. (1980). On the graphical display of the significant components of a two-way contingency table. *Communications in Statistics — Theory and Methods*, 9, 1025–1041.

Friendly, M. (1992). Graphical methods for categorical data. *SAS User Group International Conference Proceedings*, 17, 190–200.

## See Also

graphics::mosaicplot, `DescToolsX::confusion`

Other plot.bivariate: `plotBag()`, `plotBubble()`, `plotCor()`, `plotDens2D()`, `plotHeatmap()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
tab <- table(bedrock::Pizza$driver, bedrock::Pizza$area)

# default
plotAssoc(tab)

# custom palette
plotAssoc(tab, col = pal("red-white-green", n = 100))

# with residual labels
plotAssoc(tab, labels = TRUE)

# no reordering
plotAssoc(tab, reorder = FALSE)


plotAssoc(tab,
          main = "Association Driver ~ Area",
          cutoff = 1,
          xlab="Driver", ylab="Area")

cols <- pal()[c(12, 8)]
plotAssoc(tab,
          main = "Association Driver ~ Area",
          cutoff = 1,
          col = fade(cols, 0.7), border = cols,
          reorder = TRUE, cex.axis = 0.9,
          xlab = list(labels = "Driver ",
                      col = "#5B2A45", cex = 1.1),
          ylab = NA, labels = TRUE)
```

plotBag *Create a Bagplot (Bivariate Boxplot)*

## Description

Draws a bagplot (bivariate boxplot) based on halfspace (Tukey) depth. The bag contains the innermost 50\ hull of all non-outlying points, and points outside the fence (the bag inflated by `factor`, not drawn) are flagged as outliers.

## Usage

```
plotBag(x, ...)

## S3 method for class 'formula'
plotBag(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)

## Default S3 method:
plotBag(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  factor = 3,
  eps = 1e-08,
  dither = FALSE,
  points = TRUE,
  bag = TRUE,
  loop = TRUE,
  fence = FALSE,
  out = TRUE,
  median = TRUE,
  grid = FALSE,
  box = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a numeric matrix or data frame with exactly two columns, or a formula of the form y ~ x (see the formula method). Incomplete rows are removed. |
| ... | additional graphical parameters passed to par(). |
| formula | a formula of the form y ~ x, where both variables are numeric. x is drawn on the horizontal, y on the vertical axis. |
| data | an optional data frame containing the variables in formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = Weight > 2500), as in plot.formula(). |
| na.action | a function specifying how missing values are handled. Defaults to stats::na.omit(), as the depth computation requires complete pairs. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the formula, or the name of x. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | axis labels. NULL (default) uses the variable names of the formula, resp. the column names of x (empty if it has none). |
| xlim, ylim | numeric vectors of length 2 specifying axis limits. NULL (default) uses the range of the data, extended to the fence if it is drawn. |
| factor | inflation factor for the fence (default: 3). |
| eps | numeric tolerance used in depth computation and geometry, relative to the standardized coordinates. |
| dither | logical, whether to add small noise (order eps) to the data. Default FALSE: ties are handled exactly, and dithering would only break duplicates apart and so lower their depth. |
| points, bag, loop, fence, out, median, grid, box | object-oriented control of plot elements (see Details). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp(). |

## Details

All graphical elements are controlled via an object-oriented interface: each element can be specified as TRUE, FALSE, or a list(...) of graphical parameters. Internally, this is handled via bedrock::callIf().

The construction follows Rousseeuw, Ruts and Tukey (1999):

1. The halfspace (Tukey) depth of every observation is computed using a direct port of the original Fortran routine TUKDEPTH (Rousseeuw & Ruts, 1996).
2. The Tukey median is approximated by the mean of all observations with maximal depth.
3. The *bag* is obtained by radial interpolation between the convex hulls of two adjacent depth regions, calibrated such that it contains $\lfloor n/2 \rfloor$ observations (up to ties on the polygon boundary).

4. The *fence* is the bag inflated by `factor` relative to the Tukey median. Following the original proposal it is used for classification only and not drawn by default.
5. Observations outside the fence are flagged as *outliers*.
6. The *loop* is the convex hull of all non-outlying observations, so it always lies within the data range.

Two approximations remain relative to the strict theory: the Tukey median is not computed via `HALFMED`, and the bag interpolates hulls of sample depth regions rather than exact isodepth contours. Borderline outlier classifications may therefore differ slightly from other implementations (e.g. **aplpack**).

The depth computation handles exact ties and collinear configurations (e.g. discrete data) exactly; no dithering is needed. All computations run on standardized coordinates, so the result does not depend on the units of measurement.

## Value

Invisibly returns a list of class `"bagplot"` with components:

- `center` - Tukey median.
- `depth` - depth of the innermost region bounding the bag.
- `bag` - bag polygon (matrix).
- `fence` - fence polygon (matrix, not drawn by default).
- `loop` - loop polygon (matrix).
- `outliers` - outlier points (matrix).
- `depths` - halfspace depth of all observations.

## Element Control

Each of the following arguments accepts:

- `TRUE` - draw element with defaults
- `FALSE` - suppress element
- `list(...)` - customize graphical parameters

Supported elements (in drawing order):

- `grid` - background grid
- `fence` - classification boundary (default `FALSE`)
- `loop` - convex hull of the non-outlying points
- `bag` - central 50\
- `points` - raw data points
- `out` - outliers
- `median` - Tukey median
- `box` - plot frame

## Note

The halfspace depth is computed by a C++ port of the Fortran routine TUKDEPTH (Algorithm AS 307, Rousseeuw & Ruts 1996), taken from `src/depth.f` of the archived package 'depth' by M. Genest, J.-C. Massé and J.-F. Plante.

## References

Rousseeuw, P. J., Ruts, I., & Tukey, J. W. (1999). The bagplot: A bivariate boxplot. *The American Statistician*, 53(4), 382–387. doi:10.1080/00031305.1999.10474494

Rousseeuw, P. J., & Ruts, I. (1996). Algorithm AS 307: Bivariate location depth. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, 45(4), 516–526. doi:10.2307/2986073

## See Also

Other plot.bivariate: `plotAssoc()`, `plotBubble()`, `plotCor()`, `plotDens2D()`, `plotHeatmap()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
set.seed(1)
x <- cbind(rnorm(200), rnorm(200))

# data of Rousseeuw et al. (1999): car weight vs engine displacement
cardata <- data.frame(
  Weight = c(2560, 2345, 1845, 2260, 2440,
             2285, 2275, 2350, 2295, 1900, 2390, 2075, 2330, 3320, 2885,
             3310, 2695, 2170, 2710, 2775, 2840, 2485, 2670, 2640, 2655,
             3065, 2750, 2920, 2780, 2745, 3110, 2920, 2645, 2575, 2935,
             2920, 2985, 3265, 2880, 2975, 3450, 3145, 3190, 3610, 2885,
             3480, 3200, 2765, 3220, 3480, 3325, 3855, 3850, 3195, 3735,
             3665, 3735, 3415, 3185, 3690),
  Disp = c(97, 114, 81, 91, 113, 97, 97,
           98, 109, 73, 97, 89, 109, 305, 153, 302, 133, 97, 125, 146,
           107, 109, 121, 151, 133, 181, 141, 132, 133, 122, 181, 146,
           151, 116, 135, 122, 141, 163, 151, 153, 202, 180, 182, 232,
           143, 180, 180, 151, 189, 180, 231, 305, 302, 151, 202, 182,
           181, 143, 146, 146)
)

plotBag(x)

# Custom styling
plotBag(x,
  bag = list(col = adjustcolor("green", 0.3), border = "darkgreen"),
  loop = list(border = "black", lty = 3),
  out = list(col = "red", pch = 17),
  grid = TRUE
)

# Minimal plot
plotBag(x, points = FALSE, median = FALSE)
```

```
# formula interface
plotBag(Disp ~ Weight, data = cardata)

# subset, evaluated in data
plotBag(Disp ~ Weight, data = cardata, subset = Weight > 2500)

# example of Rousseeuw et al. (1999): car weight vs engine displacement;
# the axis labels default to the column names
plotBag(cardata)
```

---

plotBar *Themed Barplot with Grid, Labels and Optional Connecting Lines*

---

## Description

Creates a themed wrapper around graphics::barplot() with support for consistent styling, optional grid lines, value labels, and connecting lines for stacked barplots.

## Usage

```
plotBar(
  height,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  yax = NULL,
  beside = FALSE,
  horiz = FALSE,
  col = .useTheme,
  border = .useTheme,
  grid = .useTheme,
  box = FALSE,
  text = NULL,
  legend = TRUE,
  connLines = NULL,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| height | A vector or matrix of bar heights passed directly to graphics::barplot(). |
| main | main title. NULL (default) draws none. "", NA or FALSE also suppress it; the top margin is compacted whenever no title is drawn. |
| xlab, ylab | optional axis labels. Defaults follow base graphics behaviour. |

| | |
|---|---|
| yax | controls drawing of the numeric axis.<br>Supported values are<br>TRUE draw axis using package defaults<br>FALSE suppress axis<br>NULL do not draw a numeric axis at all<br>list(...) custom axis parameters passed to the axis drawing routine |
| beside | logical. If TRUE, bars are drawn side-by-side. If FALSE (default), bars are stacked. |
| horiz | logical. If TRUE, bars are drawn horizontally. Defaults to FALSE. |
| col | bar fill colours. .useTheme (default) resolves to getTheme()$bar$col for a vector. For a matrix with several rows, whose rows are the segments (stacked) or the bars within a group (beside = TRUE), it becomes a ramp from a dark to a light shade of that color, one per row, so the rows can be told apart. |
| border | bar border colour. .useTheme (default) resolves to getTheme()$bar$border. |
| grid | controls drawing of grid lines. Can be:<br>• .useTheme (default): follow the active theme (getTheme()$grid), restricted to the axis perpendicular to the value axis (e.g. horizontal lines only for vertical bars)<br>• TRUE: draw grid with theme settings<br>• FALSE, NULL, or NA: suppress grid<br>• a named list: arguments passed to graphics::grid(), overriding the theme/function defaults for this call only |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for grid. |
| text | optional list of arguments passed to barText() to draw value labels on bars. |
| legend | controls the legend explaining the rows of a matrix. TRUE (default) draws one whenever height is a matrix with several named rows; it is placed in the right margin, which is widened for it unless mar in ... sets the right side. FALSE, NULL or NA suppress it. A named list overrides arguments of graphics::legend(); giving x there places the legend inside the plot instead (e.g. list(x = "topleft")). Not drawn when legend.text is passed on to graphics::barplot(). |
| connLines | optional list of arguments controlling connecting lines between stacked bars. Only supported when beside = FALSE. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, or an explicit string, as for .withGraphicsState() (internal). |
| ... | additional arguments passed to graphics::barplot() and graphical parameters (via graphics::par()). |

## Details

The function first initializes the plotting region invisibly using graphics::barplot(), optionally adds grid lines, and then draws the actual bars and additional layers (axis, connecting lines, text labels).

The function internally performs the following steps:

1. Draws an invisible `barplot` to establish the coordinate system.
2. Optionally adds grid lines.
3. Draws the bars.
4. Draws the numeric axis if enabled.
5. Optionally adds connecting lines for stacked bars.
6. Optionally adds value labels via `barText()`.
7. Optionally draws a box around the plot region.

Graphical parameters such as `bg`, `cex`, `las`, `mar`, etc. can be supplied via `...`.

The precedence of theme-aware settings (`col`, `border`, `grid`, `box`, `stamp`) is

```
explicit argument  >  function-specific default  >  active theme (getTheme())
```

## Value

Invisibly returns the midpoints of the bars as returned by `graphics::barplot()`.

## See Also

graphics::barplot, barText, theme

Other plot.univariate: `plotArea()`, `plotBox()`, `plotCatDist()`, `plotDens()`, `plotDensBox()`, `plotDot()`, `plotECDF()`, `plotFreqDist()`, `plotLines()`, `plotQQ()`, `plotRidge()`, `plotViolin()`

## Examples

```
# Simple barplot
plotBar(1:5)

# With grid lines
plotBar(1:5, grid = TRUE)

# Stacked bars with labels and connecting lines
m <- matrix(c(3,2,4,1,5,2), nrow = 2)
plotBar(m,
        text = list(pos = "mid"),
        connLines = list(col = "black"))

# a matrix: one shade per row, with a legend
plotBar(VADeaths)

# Grouped bars
plotBar(VADeaths,
        beside = TRUE,
        col = gray.colors(nrow(VADeaths)))

# Horizontal bars
plotBar(VADeaths,
        horiz = TRUE,
        las = 1)
```

```
plotBar(VADeaths, ylim=c(0,250),
        grid=list(col = "grey", lty="dotted"),
        las=1, main="MyTitle",
        text = list(labels=VADeaths,
        border = NA, srt=45, bg="navajowhite"))

plotBar(VADeaths, ylim=c(0,80),
        las=1, main="MyTitle",
        box=FALSE,
        col=gray.colors(nrow(VADeaths)),
        beside=TRUE,
        text = list(col="red", bg=addOpacity("white", 0.7), border=NA))

plotBar(VADeaths, connLines = list(lwd=1, col="blue"),
        box=FALSE, las=1, main="Connecting Lines")

ptab <- proportions(VADeaths, margin=2)
plotBar(ptab,
        las=1, main="VADeaths in %",
        box=FALSE, horiz=TRUE,
        col=(cols <- gray.colors(nrow(VADeaths))),
        beside=FALSE,
        text = list(labels=fm(ptab, fmt="%"), border=NA,
                    col=contrastCol(cols)))

plotBar(VADeaths/1e3,  box=FALSE, bg="lightyellow", main="VADeaths",
        horiz=TRUE,
        text=list(border=FALSE, cex=0.8, col=c("blue", "green","orange")),
        mar=c(right=5),
        yax = list(fmt="%", d=0, big=",",
                   col="red", col.axis="blue", lwd=2))
```

---

plotBinaryTree *Binary Tree*

---

## Description

Create a binary tree of a given number of nodes n. Can be used to organize a sorted numeric vector as a binary tree.

## Usage

```
plotBinaryTree(
  x,
  main = "Binary tree",
  horiz = FALSE,
  text = TRUE,
```

```
  line = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector to be organized as binary tree. |
| main | main title of the plot. |
| horiz | logical, should the plot be oriented horizontally or vertically (default). |
| text | properties of the text, can be any of the arguments of boxedText() (besides geometry and label). |
| line | properties of the line segments (col, lwd, lty). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | the dots are sent to canvas(). |

## Details

If we index the nodes of the tree as 1 for the top, 2–3 for the next horizontal row, 4–7 for the next, ... then the parent-child traversal becomes particularly easy. The basic idea is that the rows of the tree start at indices 1, 2, 4, ....

binaryTree(13) yields the vector c(8, 4, 9, 2, 10, 5, 11, 1, 12, 6, 13, 3, 7) meaning that the smallest element will be in position 8 of the tree, the next smallest in position 4, etc.

## Value

an integer vector of length n

## Note

Substantially based on code by Terry Therneau, with major extensions and improvements by the package author.

## See Also

Other plot.special: plotCirc(), plotLift(), plotMiss(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap(), plotWeb()

## Examples

```
bedrock::binaryTree(12)

x <- sort(sample(100, 24))
z <- plotBinaryTree(x, cex=0.8)

plotBinaryTree(LETTERS[1:15],
               text=list(col="deeppink4", cex=2),
```

```
                line=list(col="navajowhite3", lwd=2))


# Plot example - Titanic data, for once from a somwhat different perspective
tab <- apply(Titanic, c(2,3,4), sum)
cprob <- c(1, prop.table(apply(tab, 1, sum))
           , as.vector(aperm(prop.table(apply(tab, c(1,2), sum), 1), c(2, 1)))
           , as.vector(aperm(prop.table(tab, c(1,2)), c(3,2,1)))
)

plotBinaryTree(round(cprob[bedrock::binaryTree(length(cprob))],2), horiz=TRUE, cex=0.8,
            main="Titanic")
text(c("sex","age","survived"), y=0, x=c(1,2,3)+1)
```

---

plotBox *Grouped Boxplot*

---

## Description

Draws boxplots for a numeric variable, optionally grouped by a categorical variable. Group means and a reference line for the overall mean can optionally be overlaid.

## Usage

```
plotBox(x, ...)

## Default S3 method:
plotBox(
  x,
  groups = NULL,
  order = NULL,
  horiz = FALSE,
  main = NULL,
  xlab = "",
  ylab = "",
  ylim = NULL,
  col = NULL,
  grid = TRUE,
  means = TRUE,
  stamp = .useTheme,
  ...
)

## S3 method for class 'formula'
plotBox(
  formula,
  data,
```

```
  subset,
  na.action = na.omit,
  horiz = FALSE,
  main = NULL,
  xlab = "",
  ylab = "",
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector, or a formula of the form x ~ g. |
| ... | graphical parameters. Parameters recognized by the internal graphics framework are applied via par(); remaining arguments are forwarded to graphics::boxplot(). By default the boxes are drawn without an outline (boxlty = 0) and with a grey median (medcol = "grey50"). Supplying border draws the outline in that color, and the median follows it; boxlty and medcol given explicitly take precedence. |
| groups | optional grouping variable (ignored if a formula is used). |
| order | optional order of the groups along the group axis: a character vector of group levels, e.g. order = c("versicolor", "virginica", "setosa"), or their positions (order = c(2, 3, 1)). Levels not mentioned follow in their original order. NULL (default) keeps the order of the factor levels. Colors given in col apply to the groups in the order drawn. |
| horiz | logical. If TRUE, the boxes are drawn horizontally, as with boxplot(horizontal = TRUE): groups on the y-axis, values on the x-axis. Default FALSE. The left margin is then widened to the longest group label, unless mar in ... sets the left side. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the formula, or x ~ g for the default method. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | label for the x-axis, i.e. the horizontal axis as drawn (the value axis if horiz = TRUE). |
| ylab | label for the y-axis, i.e. the vertical axis as drawn (the group axis if horiz = TRUE). |
| ylim | numeric vector of length 2 specifying the limits of the value axis (the x-axis if horiz = TRUE, as in graphics::boxplot()). If NULL (default), the range of x is used. |
| col | vector of fill colors, recycled over the groups. NULL (default) uses "grey90". |
| grid | controls drawing of the background grid. Can be:<br>• TRUE: draw grid with default settings (lines along the value axis only)<br>• FALSE, NULL, or NA: suppress grid<br>• a named list: arguments passed to graphics::grid(), e.g. list(col = "red", nx = NA, ny = NULL) for horizontal lines only |

| | |
|---|---|
| means | controls drawing of group means and an overall mean reference line. The reference line is drawn only for more than one group; with a single box it would merely repeat its mean. Can be:<br>• TRUE: draw with default settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to the internal means function. Supported arguments: col, pch, cex, lcol, lty, lwd. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, or an explicit string, as for .withGraphicsState() (internal). |
| formula | a formula of the form y ~ group, or y ~ a:b for the cells of several grouping variables. y ~ a + b is not accepted, see bedrock::resolveFormula(). |
| data | an optional data frame containing the variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot(). |
| na.action | a function specifying how missing values are handled, defaults to na.omit(). |

## Details

Optional plot components are controlled using bedrock::callIf() semantics:

- TRUE: draw with defaults
- FALSE, NULL, or NA: suppress component
- named list: customize component arguments

## Value

Invisibly returns NULL.

## See Also

graphics::boxplot(), bedrock::callIf()

Other plot.univariate: plotArea(), plotBar(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
set.seed(1)
x <- rnorm(100)
g <- sample(c("A", "B"), 100, TRUE)

plotBox(x)
plotBox(x, g)

plotBox(x ~ g)

# data, subset and the cells of two grouping variables
```

```
plotBox(len ~ supp, ToothGrowth, subset = dose > 0.5)
plotBox(len ~ supp:dose, ToothGrowth)

# horizontal boxes
plotBox(len ~ supp:dose, ToothGrowth, horiz = TRUE)

# groups in a chosen order, e.g. by median
plotBox(Sepal.Length ~ Species, data = iris,
        order = c("versicolor", "virginica", "setosa"))
med <- tapply(iris$Sepal.Length, iris$Species, median)
plotBox(Sepal.Length ~ Species, data = iris, order = order(med))
```

---

plotBubble *Bubble Plot*

---

## Description

Draws a bubble plot where the position is given by x and y, and the size of each bubble increases with area.

## Usage

```
plotBubble(x, ...)

## Default S3 method:
plotBubble(
  x,
  y,
  area,
  main = "",
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  add = FALSE,
  col = NA,
  border = NULL,
  cex = 1,
  grid = NA,
  na.rm = FALSE,
  stamp = .useTheme,
  ...
)

## S3 method for class 'formula'
plotBubble(
  formula,
```

```
  data = NULL,
  subset,
  na.action = na.omit,
  main = "",
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  add = FALSE,
  col = NA,
  border = NULL,
  cex = 1,
  grid = NA,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of x positions. |
| ... | additional graphical parameters passed to par(). |
| y | numeric vector of y positions. |
| area | non-negative numeric vector controlling the bubble sizes. |
| main, xlab, ylab | plot labels. |
| xlim, ylim | axis limits. |
| add | logical; if TRUE, adds to an existing plot. |
| col | fill color(s) of the bubbles. |
| border | border color(s) of the bubbles. |
| cex | scaling factor applied to the bubble radii. |
| grid | logical, NA, or list controlling background grid. |
| na.rm | logical. Observations with a missing x, y or area cannot be drawn and are always omitted; with na.rm = FALSE (default) a warning reports how many. Missing colors are not missing data, NA draws transparent. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| formula | A formula of the form y ~ x \| area. |
| data | optional data frame. |
| subset | optional subset expression, evaluated in data. |
| na.action | function to handle missing values, defaults to na.omit(). |

## Details

The function supports both a default interface and a formula interface of the form y ~ x | area.

The bubble radii are scaled relative to the largest area: the largest bubble gets a radius of 0.35 * cex inches, the others $r = 0.35 \cdot cex \cdot (area/\max(area))^{0.4}$. The exponent 0.4 (instead of 0.5 for

strictly area-proportional circles) dampens the size differences, so that small bubbles remain visible. Radii are converted to user coordinates separately for both axes, so the bubbles are drawn as circles regardless of the aspect ratio.

Bubbles are drawn in order of decreasing area, so small bubbles are not hidden behind large ones.

Graphical elements such as grids are controlled via the unified plot design system using `bedrock::callIf()` and `.theme()`.

## Value

Invisibly, a data frame with one row per drawn bubble and the columns `x`, `y`, `area`, `col`, `border`, and the radii `rx`, `ry` in user coordinates, sorted by decreasing area.

## See Also

`symbols()`

Other plot.bivariate: `plotAssoc()`, `plotBag()`, `plotCor()`, `plotDens2D()`, `plotHeatmap()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
set.seed(1)
x <- rnorm(100)
y <- rnorm(100)
a <- runif(100, 1, 10)

plotBubble(x, y, a)

df <- data.frame(x = x, y = y, a = a)
plotBubble(y ~ x | a, data = df)

US <- data.frame(state.x77, State=state.name,
                   Region=state.region, Abb=state.abb)
plotBubble(Income ~ Population | Area ,
           data=US,
           grid=TRUE, col=addOpacity(pal("helsana")[US$Region]), cex=1.2 )

text(Income ~ Population, US, labels=US$Abb, cex=0.8)
```

---

# plotCatDist *Categorical Distribution Plot*

---

## Description

Visualizes the distribution of a categorical variable using horizontal bar plots of absolute and relative frequencies. Optionally, cumulative proportions (ECDF-style) can be displayed.

## Usage

```
plotCatDist(
  x,
  main = NULL,
  type = c("both", "freq", "perc"),
  ecdf = FALSE,
  maxCats = NULL,
  maxLabLen = 25,
  col = "grey80",
  border = FALSE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a factor or character vector, a one-dimensional table of counts, or a numeric vector of precomputed frequencies. Unnamed numeric frequency vectors are labelled by their positions. |
| main | plot title. |
| type | character; one of "both", "freq", "perc". Controls whether absolute frequencies, relative frequencies, or both are displayed. |
| ecdf | logical; if TRUE, cumulative proportions are shown instead of simple relative frequencies. |
| maxCats | optional maximum number of categories to display (truncates if exceeded). |
| maxLabLen | integer; maximum length of category labels before truncation. |
| col | fill color for bars. |
| border | logical; draw borders around bars. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to par(). |

## Details

The function produces horizontal bar plots:

- Absolute frequencies (counts)
- Relative frequencies (percentages) or cumulative proportions

If type = "both", both views are shown side by side.

Long labels are truncated, and large category sets can be limited via maxCats.

Raw categorical data and their pre-tabulated form are treated identically. When categories are truncated via maxCats, proportions remain based on the total frequency before truncation.

## Value

Invisibly returns a list with frequencies and proportions.

## See Also

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
# Basic usage
x <- factor(sample(letters[1:5], 100, TRUE))
plotCatDist(x)
plotCatDist(table(x))

# Only proportions
plotCatDist(x, type = "perc")

# With cumulative distribution
plotCatDist(x, ecdf = TRUE)

# Many categories (truncation)
x2 <- factor(sample(letters, 200, TRUE))
plotCatDist(x2, maxCats = 10)
```

---

# plotCirc *Circular Chord Diagram*

---

## Description

Draws a chord diagram of a two-way table: the categories of the rows lie as sectors on the right half of a circle, those of the columns on the left half, and every cell of the table is a ribbon between its row and its column sector whose width is proportional to its value.

## Usage

```
plotCirc(
  x,
  main = NULL,
  gap = 3,
  opacity = 0.5,
  sector = TRUE,
  ribbon = TRUE,
  labels = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a numeric matrix or two-way table with non-negative entries. Rows and columns are the two sets of categories; the dimnames give the labels. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then the name of x. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| gap | gap between two sectors, in degrees. |
| opacity | opacity of the default ribbon colors, between 0 and 1. |
| sector | sector styling. Can be:<br>• TRUE (default): the qualitative palette of the active theme (see theme), without border,<br>• FALSE or NULL: no sectors,<br>• a vector of colors,<br>• a list with the elements col and/or border.<br>Colors are given for the rows followed by the columns of x and are recycled to their number. |
| ribbon | ribbon styling. Can be:<br>• TRUE (default): the color of the row sector a ribbon starts from, with the opacity opacity, without border,<br>• FALSE or NULL: no ribbons,<br>• a vector of colors, one per row of x (recycled),<br>• a list with the elements col and/or border. |
| labels | label styling. Can be:<br>• TRUE (default): the row and column names of x, written horizontally,<br>• FALSE or NULL: no labels,<br>• a character vector of labels, for the rows followed by the columns of x,<br>• a list with the elements labels, cex, col and las; las = 2 writes the labels radially, perpendicular to the circle. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to par() via the internal framework. |

## Details

The size of a sector is the margin total of its category. Within a row sector the ribbons are ordered like the column sectors they lead to, within a column sector like the row sectors, both from top to bottom; this keeps crossings few. Empty cells draw no ribbon.

The plot region is enlarged as far as needed for the labels, so that long category names are not cut off.

## Value

Invisibly, a list with the data frame sectors (category, margin ("row" or "col"), total, start and end angle in radians, color) and the label positions x and y.

## See Also

Other plot.special: plotBinaryTree(), plotLift(), plotMiss(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap(), plotWeb()

## Examples

```
set.seed(1)
x <- matrix(sample(1:10, 36, replace = TRUE), nrow = 6,
            dimnames = list(LETTERS[1:6], letters[1:6]))

plotCirc(x)

# a contingency table, radial labels
plotCirc(HairEyeColor[, , "Female"], labels = list(las = 2))

# own colors and white borders
plotCirc(x,
         sector = list(col = hcol(), border = "white"),
         ribbon = list(border = "white"))
```

---

# plotCor *Correlation Matrix Plot with Theming and Optional Labels*

---

## Description

Draws a correlation matrix using graphics::image() with optional clustering, triangular display, grid lines, color legend, and numeric labels inside the cells.

## Usage

```
plotCor(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xax = TRUE,
  yax = TRUE,
  minCor = 0,
  cluster = FALSE,
  triangle = c("full", "upper", "lower"),
  diag = TRUE,
  col = .useTheme,
  grid = .useTheme,
  box = .useTheme,
  legend = TRUE,
  text = FALSE,
  stamp = .useTheme,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| x | A numeric correlation matrix. |
| main, xlab, ylab | optional plot labels. |
| xax, yax | controls drawing of the axes.<br>Supported values are<br>TRUE draw axis using default settings<br>FALSE suppress axis<br>list(...) custom axis parameters passed to graphics::axis() |
| minCor | numeric threshold; correlations with absolute value smaller than this are suppressed (set to NA). |
| cluster | logical; if TRUE, variables are reordered by hierarchical clustering to place similar correlations together. |
| triangle | which part of the matrix to display. One of "full", "upper", or "lower". |
| diag | logical; should the diagonal be displayed. |
| col | color palette used for the correlation values. .useTheme (default) builds a diverging ramp from getTheme()$twin - the active theme's two-color pair - through white: twin[1] at the negative end ($-1$), white at zero, twin[2] at the positive end ($+1$). |
| grid | controls drawing of cell-separator grid lines at the half-integer matrix boundaries (clipped to the matrix extent, so they never bleed into the margins). Can be:<br>• .useTheme (default): follow the active theme (getTheme()$grid$col/$lwd)<br>• TRUE: draw with theme settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: override col/lwd for this call only |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for grid. |
| legend | logical; draw a color legend for the correlation scale. |
| text | controls numeric labels drawn inside the matrix cells.<br>Supported values are<br>FALSE no labels<br>TRUE default labels based on the correlation values<br>list(...) custom parameters passed to the internal text drawing routine |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to graphics::par() and graphics::image(). |

## Details

The function follows the DescToolsX plotting conventions:

- User arguments override theme settings.
- Theme settings override base graphics defaults.
- Graphical parameters (e.g. `cex`, `las`, `mar`) can be supplied via `...`.

The function internally:

1. Optionally reorders the matrix using hierarchical clustering.
2. Masks parts of the matrix according to `triangle` and `diag`.
3. Adjusts plot margins based on label sizes.
4. Draws the matrix using `graphics::image()`.
5. Optionally adds grid lines, numeric labels, axes, and a color legend.

Grid lines are drawn via clipped `graphics::abline()` calls at the matrix's half-integer cell boundaries rather than via `graphics::grid()`: `grid()`'s nx/ny divide the full plot region (`par("usr")`), which may carry axis padding unrelated to `image()`'s integer cell geometry, whereas the clipped approach stays exact regardless of that padding.

## Value

Invisibly returns the (possibly reordered) matrix used for plotting.

## See Also

`graphics::image()`, `stats::cor()`, theme

Other plot.bivariate: `plotAssoc()`, `plotBag()`, `plotBubble()`, `plotDens2D()`, `plotHeatmap()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
m <- cor(swiss)

# full correlation matrix
plotCor(m, legend=FALSE)

# upper triangle only
plotCor(m, triangle = "upper")

# clustered variables
plotCor(m, cluster = TRUE)

# with correlation values
plotCor(m, text = TRUE)

# customized labels
plotCor(m,
         text = list(col = "black", cex = 0.9))
```

```
# hide grid
plotCor(m, grid = FALSE)

plotCor(m, col=colorRampPalette(c("red", "black", "green"), space = "rgb")(20))
plotCor(m, col=colorRampPalette(c("red", "black", "green"), space = "rgb")(20),
         legend=FALSE)

m <- cor(mtcars)
plotCor(m, col=pal("red-white-blue-1", 100), border="grey",
         legend=list(labels=format(seq(-1,1,.25), digits=2), box=list(border="grey")))

# display only correlation with a value > 0.7
plotCor(m, minCor = 0.7)
x <- matrix(rep(1:ncol(m),each=ncol(m)), ncol=ncol(m))
y <- matrix(rep(ncol(m):1,ncol(m)), ncol=ncol(m))
txt <- format(m, digits=3)
idx <- upper.tri(matrix(x, ncol=ncol(m)), diag=FALSE)

# place the text on the upper triagonal matrix
text(x=x[idx], y=y[idx], label=txt[idx], cex=0.8, xpd=TRUE)

# put similiar correlations together
plotCor(m, cluster=TRUE)

# same as
idx <- order.dendrogram(as.dendrogram(
          hclust(dist(m), method = "mcquitty")
       ))
plotCor(m[idx, idx])

# plot only upper triangular matrix and move legend to bottom
m <- cor(mtcars)
m[lower.tri(m, diag=TRUE)] <- NA

# get the p-values
p <- outer(
  (vars <- colnames(mtcars)), vars,
  Vectorize(function(v1, v2)
    cor.test(mtcars[[v1]], mtcars[[v2]], method = "pearson")$p.value
  )
)
dimnames(p) <- list(vars, vars)
m[p > 0.05] <- NA

plotCor(m, mar=c(8,8,8,8), yaxt="n",
         legend = list(x="bottom", inset=-.15, horiz=TRUE,
                       height=abs(lineToUser(line = 2.5, side = 1)),
                       width=ncol(m)))
mtext(text = rev(rownames(m)), side = 4, at=1:ncol(m), las=1, line = -5, cex=0.8)
```

plotDens *Grouped Density Plot*

## Description

Draws kernel density estimates for one or more groups. Supports both classical density plots and conditional density plots.

## Usage

```
plotDens(x, ...)

## Default S3 method:
plotDens(
  x,
  ...,
  main = NULL,
  xlab = "",
  ylab = "density",
  xlim = NULL,
  ylim = NULL,
  bw = "nrd0",
  add = FALSE,
  col = NULL,
  lwd = 2,
  lty = 1,
  fill = FALSE,
  grid = NULL,
  legend = TRUE,
  stamp = .useTheme
)

## S3 method for class 'formula'
plotDens(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = "",
  ylab = NULL,
  type = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A numeric vector or list of numeric vectors. |

| | |
|---|---|
| ... | additional data vectors (unnamed, default method) or graphical parameters passed to par(). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the names of the data arguments, or the formula. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | axis labels. |
| xlim, ylim | axis limits. NULL (default) uses the range of the densities, the y-axis starting at 0 (c(0, 1) for a conditional density). |
| bw | bandwidth passed to stats::density() or cdplot. |
| add | logical; if TRUE, adds to an existing plot. |
| col | line color(s). |
| lwd | line width(s). |
| lty | line type(s). |
| fill | for type = "density": FALSE (default, no fill), TRUE (translucent fill derived from each group's col via adjustcolor(col, alpha.f = 0.3)), or one or more explicit fill colors recycled over groups. For type = "conditional" on a single, unstratified, binary curve: TRUE for cdplot-style grey shading, or a vector of 2 colors for the regions below/above the boundary curve. |
| grid | logical, NA, or list controlling background grid. |
| legend | controls the legend, drawn only for more than one curve, with the group names (the level names for the formula method, the names of the data arguments otherwise). TRUE (default) draws it with default settings, FALSE/NULL/NA suppress it, a named list is passed on to graphics::legend() (e.g. list(x = "topleft")). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, or an explicit string, as for .withGraphicsState() (internal). |
| formula | a formula of the form y ~ group (or y ~ a:b for the cells of several grouping variables), y ~ x (x numeric, conditional density), or y ~ x \| group. y ~ a + b is not accepted, see bedrock::resolveFormula(). |
| data | an optional data frame containing the variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot(). |
| na.action | a function specifying how missing values are handled, defaults to na.omit(). |
| type | character string specifying the plot type. One of "density", "conditional", or NULL (default, determined by resolveFormula()'s design classification). |

## Details

The function defers entirely to bedrock::resolveFormula()'s design classification to pick a mode when type = NULL:

- y ~ g (g categorical) → density, one curve per group.

- y ~ x (x numeric) → conditional density $P(Y|X)$, a single curve - equivalent to `cdplot(x, factor(y))`.
- y ~ x | g → conditional density, one curve per level of g.

`type` can be set explicitly to override the default for a given design (e.g. to force an error rather than silently doing the wrong thing if a formula's shape is ambiguous).

Graphical elements such as grids are controlled via the unified plot design system using `bedrock::callIf()` and `.theme()`.

## Value

Invisibly returns `NULL`.

## See Also

`stats::density()`, `graphics::cdplot()`, `bedrock::resolveFormula()`

Other plot.univariate: `plotArea()`, `plotBar()`, `plotBox()`, `plotCatDist()`, `plotDensBox()`, `plotDot()`, `plotECDF()`, `plotFreqDist()`, `plotLines()`, `plotQQ()`, `plotRidge()`, `plotViolin()`

## Examples

```
set.seed(1)
x <- rnorm(100)
g <- rep(c("A", "B"), each = 50)

# standard density (k = 2 groups)
plotDens(x ~ g)

# conditional density, single curve - auto-detected, no type= needed
y <- rbinom(100, 1, plogis(x))
plotDens(y ~ x)

# same, with cdplot-style fill
plotDens(y ~ x, fill = c("red", "blue"))

# conditional density, stratified by group
plotDens(y ~ x | g)

# subset and the cells of two grouping variables
plotDens(len ~ supp:dose, ToothGrowth, subset = dose > 0.5)
```

---

# plotDens2D *Two-Dimensional Kernel Density Plot*

---

## Description

Computes a two-dimensional kernel density estimate and visualises it using contour, image, or perspective plots.

## Usage

```
plotDens2D(
  x,
  y,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  type = c("contour", "image", "persp"),
  col = rev(pal("red-black", n = 100)),
  grid = .useTheme,
  box = .useTheme,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of x-coordinates. |
| y | numeric vector of y-coordinates. Must have the same length as x. |
| main | optional main title of the plot. |
| xlab, ylab | axis labels. |
| xlim, ylim | numeric vectors of length two specifying axis limits. |
| type | character string specifying the plot type. One of "contour", "image", or "persp". |
| col | color specification used for type = "image". Defaults to a reversed "red-black" sequential ramp (pal()), running from black (low density) to red (high density) - hardcoded rather than theme-driven, since this is a continuous, unidirectional gradient, unlike the active theme's categorical palette or diverging twin pair, neither of which fits a density surface. |
| grid | controls drawing of the background grid. .useTheme (default) follows the active theme (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for graphics::box(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to underlying plotting functions. |

## Details

The function estimates a bivariate density surface using a Gaussian kernel with bandwidths determined via a normal reference rule. The density is evaluated on a regular grid and visualised using one of three base graphics representations:

- "contour": contour lines of equal density

- `"image"`: raster representation of the density surface
- `"persp"`: three-dimensional perspective plot

The choice of representation affects interpretability: contour and image plots emphasise structure in the data distribution, while perspective plots highlight global shape but may distort local density.

Bandwidth selection follows a rule-of-thumb approach based on spread (interquartile range and variance), which provides a reasonable default for unimodal distributions but may oversmooth multimodal structures.

Missing or non-finite values are not allowed and will result in an error.

## Value

Invisibly returns the result of the selected plotting call. Typically a list containing grid coordinates and estimated density values.

## See Also

Other plot.bivariate: `plotAssoc()`, `plotBag()`, `plotBubble()`, `plotCor()`, `plotHeatmap()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
set.seed(1)
x <- rnorm(200)
y <- x + rnorm(200)

plotDens2D(x, y)

plotDens2D(x, y, type = "image")
```

---

# plotDensBox *Density and Boxplot Combination (Grouped)*

---

## Description

Combines density plots and horizontal boxplots for a numeric variable, optionally grouped by a categorical variable. The density plot shows the distribution shape, while the boxplot summarizes key statistics such as median, spread, and outliers.

## Usage

```
plotDensBox(x, ...)

## Default S3 method:
plotDensBox(
  x,
```

```
  groups = NULL,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  heights = c(2, 1.4),
  col = NULL,
  grid = TRUE,
  dens = TRUE,
  box = TRUE,
  stamp = .useTheme,
  ...
)

## S3 method for class 'formula'
plotDensBox(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = "",
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector, or a formula of the form x ~ g. |
| ... | further graphical parameters passed to graphics::par() via the internal framework. |
| groups | optional grouping variable (ignored if a formula is used). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the formula, or x resp. x ~ g for the default method. "", NA or FALSE always suppress it and remove its outer margin; any other string is used as given. |
| xlab | label for the x-axis, drawn below the boxplot. For the formula method, an empty label defaults to the response. |
| ylab | label for the y-axis of the density panel. |
| xlim | numeric vector of length 2 specifying the x-axis limits. NULL (default) covers the data and the tails of the densities. |
| heights | numeric vector of length 2 specifying the relative heights of the density plot (top) and boxplot (bottom). |
| col | vector of colors, recycled over the groups. If NULL, a palette is generated. |
| grid | controls drawing of the background grid. Can be:<br>• TRUE: draw grid with default settings<br>• FALSE, NULL, NA: suppress grid |

- a named list: arguments passed to graphics::grid()

dens — controls density estimation via stats::density(). Can be:

- TRUE: use default density settings
- FALSE, NULL, NA: suppress densities
- a named list: additional arguments passed to stats::density()

box — controls drawing of boxplots via graphics::boxplot(). Can be:

- TRUE: use default boxplot settings
- FALSE, NULL, NA: suppress boxplots
- a named list: additional arguments passed to graphics::boxplot()

stamp — controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp().

formula — a formula of the form y ~ group, or y ~ a:b for the cells of several grouping variables. y ~ a + b is not accepted, see bedrock::resolveFormula().

data — an optional data frame containing the variables in the formula.

subset — an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot().

na.action — a function specifying how missing values are handled, defaults to na.omit().

## Details

The function arranges two plots vertically using layout(): a density plot on top and a horizontal boxplot below. When a grouping variable is provided, densities and boxplots are drawn for each group.

Optional plot components are controlled using bedrock::callIf() semantics:

- TRUE: draw with defaults
- FALSE: suppress component
- named list: customize component arguments

## Value

Invisibly returns NULL.

## See Also

stats::density(), graphics::boxplot(), bedrock::callIf()

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
set.seed(1)
x <- rnorm(100)
g <- sample(c("A", "B"), 100, TRUE)
```

```
plotDensBox(x)
plotDensBox(x, g)

plotDensBox(
  x,
  dens = list(adjust = 2),
  box  = list(notch = TRUE)
)

plotDensBox(
  x,
  box = FALSE
)

plotDensBox(x ~ g)

# subset and the cells of two grouping variables
plotDensBox(len ~ supp:dose, ToothGrowth, subset = dose > 0.5)
```

---

plotDot *Dot Plot for Estimates and Confidence Intervals*

---

## Description

Displays numeric estimates as points on a horizontal scale. Optional confidence limits are shown as horizontal lines with capped endpoints. Several series of estimates can be arranged in labelled groups.

## Usage

```
plotDot(
  x,
  main = NULL,
  xlim = NULL,
  xax = NULL,
  axes = TRUE,
  items = NULL,
  groups = NULL,
  gap = 1,
  pch = .useTheme,
  cex = .useTheme,
  grid = .useTheme,
  box = .useTheme,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric estimates or confidence interval data. Supported formats are a numeric vector, a numeric matrix, a three-dimensional numeric array, or a "CI" object created with as.CI() |
| main | main title. NULL (default) draws none. "", NA or FALSE also suppress it; the top margin is compacted whenever no title is drawn. |
| xlim | numeric vector containing the limits of the horizontal axis; by default, the range of all estimates and confidence limits |
| xax | optional specification for the horizontal axis, interpreted by the internal axis renderer |
| axes | logical; whether the horizontal and item axes are drawn |
| items | optional character vector containing the item labels; defaults to the row names or first dimension names of x |
| groups | optional character vector containing the group labels; defaults to the column names or third dimension names of x |
| gap | non-negative number, the vertical space between groups in item rows |
| pch | specification for the estimate points. The default .useTheme uses the point settings of the active theme. A plotting symbol or a named list containing parameters such as pch, col, bg, and cex can also be supplied |
| cex | character expansion factor for the points. .useTheme (default) resolves to getTheme()$points$cex. |
| grid | specification controlling the horizontal item and group grid lines. The default .useTheme follows the active theme. A logical value, NA, or a named list of graphical parameters can also be supplied |
| box | specification controlling the plot box. The default .useTheme uses the active theme. A logical value, NA, or a named list of graphical parameters can also be supplied |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to graphics::par(). A mar that sets the left side switches off the automatic widening of the left margin for the labels. |

## Details

A numeric vector represents one estimate for each item; a one-dimensional table or array, such as the result of tapply() with one grouping factor, is treated the same way.

A numeric matrix represents estimates only: rows define the items and columns define the groups. Consequently, a matrix with three columns is interpreted as three groups and not automatically as estimates with lower and upper confidence limits.

Use as.CI() to declare explicitly that a matrix, data frame, list, or result from tapply() contains confidence interval data:

```
plotDot(as.CI(x))
```

A "CI" object contains the columns est, lci, and uci. Additional columns can define the item and group structure. If two additional columns are present, the first defines the items and the second defines the groups.

Confidence interval data can alternatively be supplied as a three-dimensional numeric array with dimensions items × 3 × groups. The second dimension must contain, in this order, the estimate, lower confidence limit, and upper confidence limit.

Values supplied directly as arguments take precedence over the corresponding settings of the active theme. Vectors in pch (e.g. pch = list(col = c("red", "blue"))) are recycled over the groups, in the order in which the groups are drawn from top to bottom.

## Value

invisibly, a list containing:

ypos vertical positions of the items within each group, a list in the order of the groups

group_y vertical positions of the group labels

sep_y vertical positions of the group separators

## See Also

as.CI(), isCI(), graphics::dotchart()

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
# estimates for a single series
est <- c(A = 12, B = 18, C = 28, D = 40, E = 65)

plotDot(
  est,
  main = "Estimates"
)

# matrix columns represent groups of estimates
groupedEst <- cbind(
  Control = c(A = 12, B = 18, C = 28),
  Treatment = c(A = 16, B = 24, C = 35)
)

plotDot(
  groupedEst,
  main = "Grouped estimates"
)

# confidence intervals stored in a matrix
ci <- cbind(
  est = est,
  lci = est - c(2, 3, 4, 5, 6),
  uci = est + c(2, 3, 4, 5, 6)
```

```
)

plotDot(
  as.CI(ci),
  main = "Estimates with confidence intervals"
)

# grouped confidence intervals stored in a data frame
groupedCI <- data.frame(
  item = rep(c("A", "B", "C"), 2),
  group = rep(c("Control", "Treatment"), each = 3),
  estimate = c(12, 18, 28, 16, 24, 35),
  lower = c(10, 15, 24, 13, 20, 30),
  upper = c(14, 21, 32, 19, 28, 40)
)

plotDot(
  as.CI(
    groupedCI,
    estimate = "estimate",
    lower = "lower",
    upper = "upper"
  ),
  main = "Grouped confidence intervals"
)

# returned positions can be used to add graphical elements
pos <- plotDot(est)

points(
  est + 3,
  y = unlist(pos$ypos),
  pch = 4
)
```

---

plotECDF *Empirical Cumulative Distribution Function*

---

## Description

Fast plotting of the empirical cumulative distribution function (ECDF), designed to stay performant even for very large vectors (n ~ 1e6-1e7).

## Usage

```
plotECDF(x, ...)

## Default S3 method:
plotECDF(
```

```
  x,
  main = NULL,
  xlab = NULL,
  ylab = "",
  xlim = NULL,
  breaks = 1000,
  add = FALSE,
  col = .useTheme,
  lwd = 2,
  grid = .useTheme,
  box = .useTheme,
  stamp = .useTheme,
  ...
)

## S3 method for class 'formula'
plotECDF(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = NULL,
  xlim = NULL,
  col = .useTheme,
  lwd = 2,
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of the observations for the ECDF. Missing values are removed. |
| ... | further graphical parameters passed to par() via the internal framework. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the expression passed as x. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | label for the x-axis. NULL (default) derives a label the same way as main. |
| ylab | label for the y-axis. The y-axis itself always shows fixed probability labels (.00 to 1.00); ylab adds an axis title above/beside those, empty by default. |
| xlim | numeric vector of length 2; x-axis limits. NULL (default) uses range(x). |
| breaks | controls the rendered resolution. A single integer (default 1000) subsamples the ECDF at that many evenly-spaced quantiles whenever length(x) exceeds it; for length(x) <= breaks, full resolution is used regardless (no data is ever thinned below its actual size). NULL, FALSE, or Inf force full resolution unconditionally, regardless of length(x). |

| | |
|---|---|
| add | logical; if TRUE, adds to an existing plot instead of starting a new one. |
| col | color of the step line and the min/max marker points. .useTheme (default) resolves to getTheme()$twin[1] - a single accent color, consistent with lines.loess() and plotQQ()'s confidence band. |
| lwd | line width. |
| grid | controls drawing of the background grid (vertical lines at default tick positions, plus a fixed set of horizontal reference lines at the probability ticks 0/.25/.5/.75/1<br>• the latter always grey regardless of the active theme, a deliberately distinct look, not theme-driven). .useTheme (default) follows the active theme's grid on/off state (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for graphics::box(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list of arguments for stamp(). |
| formula | a formula of the form y ~ group, one ECDF per group, or y ~ a:b for the cells of several grouping variables. y ~ a + b is not accepted, see bedrock::resolveFormula(). |
| data | an optional data frame containing the variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot(). |
| na.action | a function specifying how missing values are handled. Defaults to na.omit. |
| legend | controls the legend of the formula method, drawn with the group levels. TRUE (default) draws it with default settings, FALSE/NULL/NA suppress it, a named list is passed on to graphics::legend() (e.g. list(x = "topleft")). |

## Details

The base plot.ecdf()/ecdf() machinery becomes impractically slow well below n = 1e7, since it tracks every single jump. Beyond a few thousand points, individual jumps are visually indistinguishable anyway, so plotECDF() caps the rendered resolution at breaks points by default.

Resolution is achieved via quantile subsampling, not histogram binning: breaks evenly-spaced probability points are mapped back to their corresponding quantiles (stats::quantile(), type = 7). Each rendered point therefore lies exactly on the true ECDF - unlike an equal-width-histogram approximation, this adapts automatically to the shape of the distribution (no resolution is wasted on near-empty bins in a skewed or heavy-tailed distribution, nor lost in the tails).

## Value

Invisibly returns NULL.

## See Also

plot.ecdf(), plotFreqDist(), theme

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotFreqDist(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
plotECDF(faithful$eruptions)

# large vector - automatically thinned to 1000 points, no breaks= needed
x <- rnorm(1e6)
plotECDF(x)

# force full resolution regardless of size
plotECDF(x, breaks = NULL)

# grouped ECDFs via the formula interface
plotECDF(Sepal.Length ~ Species, data = iris)

# subset and the cells of two grouping variables
plotECDF(len ~ supp:dose, ToothGrowth, subset = dose > 0.5)
```

---

# plotFacet *Facet Panel Matrix in Base Graphics*

---

## Description

Draws a lattice-like matrix of panels in base graphics with identical plot region sizes, panel strips, axes on the outer panels only and a user-supplied panel function for the content. Panel gaps are defined in margin lines and remain exact, independent of device size and strip height, as the strip space is reserved separately in the layout.

## Usage

```
plotFacet(
  samples,
  dim,
  FUN,
  main = "",
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  horiz = 1,
  vert = NULL,
  col = NULL,
  bg = "grey95",
  strip = TRUE,
  grid = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| samples | a list of samples, each a list (or data.frame) with components x and y. |
| dim | integer vector of length 2, the number of rows and columns of the panel matrix, c(nrow, ncol). |
| FUN | the panel function, called per panel as FUN(x, y, col, pch, ...) with a fully set up coordinate system. Components of a sample beyond x and y are passed on under their own names, so a panel can carry per-panel data of its own - confidence bounds, weights, labels. They are only passed to a FUN that can accept them (a matching formal, or ...), so panel functions written for the two-component form keep working unchanged. |
| main | the main title, placed in the outer margin. |
| xlab, ylab | the axis labels, placed in the outer margins. |
| xlim, ylim | the axis limits. A numeric vector of length 2 (or NULL, the default, for the range over all samples) binds every panel to the same scale. "free" gives each panel its own scale, taken from its own sample. A *list* of length 2 vectors, one per sample, does the same with limits you choose; a list of length 1 is recycled. See the section on free scales. |
| horiz | the horizontal gap between adjacent columns in margin lines. |
| vert | the vertical gap between adjacent rows in margin lines. Default is horiz, yielding physically equal gaps. |
| col | the colors for the panels, recycled to the number of samples. Default is hcl.colors(n, "Dark 3"). |
| bg | the background color of the plot regions. |
| strip | controls the panel strips: TRUE (default) draws strips with default settings, FALSE/NULL/NA suppresses them (no space is reserved), a named list sets the details. Its element labels holds the strip labels (default: names(samples) where the samples are named, otherwise the sequence along samples); all other elements are passed as arguments to titleRect(), e.g. list(labels = vars, bg = "steelblue", col = "white", line = 1.5). The label argument of titleRect() is set per panel and cannot be overridden. |
| grid | controls the grid lines, evaluated by bedrock::callIf: TRUE (default) draws grid lines at the positions of graphics::axTicks() with default settings (col = "grey85", lwd = 0.8), FALSE/NULL/NA suppresses them, a named list is passed as arguments to abline(), e.g. list(col = "white", lty = "dotted"). The default positions v and h can be overridden, e.g. list(v = seq(0, 20, 5)). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further arguments passed to FUN. Three graphical parameters are picked up here and define the page geometry instead: mar, the margins around the whole panel matrix in lines, c(bottom, left, top, right), default c(2.5, 2.5, 0.5, 0.5) (the bottom and left margins hold the axis annotation of the outer panels); oma, the outer margins in lines, holding xlab, ylab and main, default c(3, 3, 4, 1.2); and cex, the character expansion used inside the panels (axis annotation, strip labels, panel content) and as unit for the panel margin lines, default 0.66, matching R's own reduction in multi-figure layouts (set deterministically after each plot.new(), see Details). pch defaults to 16 and is passed to FUN. |

## Details

The available device area inside the outer margins is partitioned with `layout()` such that all plot regions have exactly the same size in inches. The horizontal gap between two adjacent columns is `horiz` margin lines, the vertical gap between two adjacent rows is `vert` lines. Since margin lines have the same physical size in both directions, `horiz == vert` yields visually equal gaps.

## Value

Invisibly returns a list with the realized geometry: `horiz`, `vert`, `strip_line` (reserved strip height in lines) and the common plot region size `plot_width_in`, `plot_height_in` in inches.

## Bound and free scales

By default all panels share one coordinate system, and the axes are drawn on the outer panels only - the arrangement that makes small multiples comparable at a glance.

Passing `xlim = "free"` - or a list of limits, for control over the individual panels - frees that dimension: every panel gets its own limits, and with them its own axis, because an outer axis would no longer describe the panels above or beside it. The layout answers for this - a freed dimension reserves the full `mar` on every panel edge that now carries annotation, not just on the outer ones, so the panels stay equal in size and the tick labels have room.

Free scales cost what they free: panels can no longer be compared by position, only by shape. The natural case is a set of diagnostics of one model against different predictors - the residual scale is shared and worth comparing, the predictor scales are not commensurable at all.

The strip is drawn with `titleRect()` above each panel. Its height (`line` argument of `titleRect`) is reserved in the top margin of every panel, so the strip never eats into the gap between the rows.

Note that `plot.new()` silently reduces `cex` (and with it `csi`, the physical size of a margin line) in layouts with more than two regions, which would make the realized panel margins deviate from the computed layout. The function therefore controls the character size deterministically via `cex` (see `...`) and sets the panel margins in inches (`mai`/`omi`), so that all plot regions are exactly equal in size.

## See Also

graphics::layout, titleRect, bedrock::callIf

Other graphics.layout: `abcCoords()`, `axTicksPosixct()`, `axisBreak()`, `isValidPlotRegion()`, `lineToUser()`, `mar()`, `spreadOut()`

## Examples

```
samples <- lapply(split(ChickWeight, ChickWeight$Chick)[1:25],
                  function(z) list(x = z$Time, y = z$weight))

my_panel <- function(x, y, col, pch = 16, ...) {
  points(x, y, pch = pch, col = col)
  abline(lm(y ~ x), lwd = 1)
}

plotFacet(samples, dim = c(5, 5), FUN = my_panel,
          xlab = "Time", ylab = "Weight", main = "ChickWeight",
```

```
           strip = list(bg = "grey80", cex = 0.8))

# free x scales: mpg against four predictors, each on its own range,
# with a shared y scale. The regression band travels in the samples and
# reaches the panel function under its own names.
vars <- c("disp", "hp", "wt", "qsec")

samples <- lapply(vars, function(v) {
  ord <- order(mtcars[[v]])
  ci  <- predict(lm(reformulate(v, "mpg"), mtcars), interval = "confidence")
  list(x = mtcars[[v]][ord], y = mtcars$mpg[ord],
       lci = ci[ord, "lwr"], uci = ci[ord, "upr"])
})

panelBand <- function(x, y, lci, uci, col, pch = 16, ...) {
  polygon(c(x, rev(x)), c(uci, rev(lci)), col = "grey85", border = NA)
  points(x, y, col = col, pch = pch)
}

plotFacet(setNames(samples, vars), dim = c(2, 2), FUN = panelBand,
          xlim = "free", ylab = "mpg",
          main = "mpg against four predictors")
```

---

plotFreqDist *Frequency Distribution Plot*

---

## Description

Creates a univariate graphical summary combining a histogram (or probability mass plot for discrete data), a kernel density curve, a boxplot, and an empirical cumulative distribution function in a single multi-panel figure. Optional components include a rug, and fitted theoretical distribution curves for both the histogram and the ECDF panel.

## Usage

```
plotFreqDist(
  x,
  main = NULL,
  xlab = "",
  xlim = NULL,
  heights = NULL,
  hist = TRUE,
  dens = TRUE,
  rug = FALSE,
  curve = FALSE,
  boxplot = TRUE,
  ecdf = TRUE,
  curveEcdf = FALSE,
```

```
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector whose distribution is to be plotted. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the expression passed as x. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | label for the x-axis. The variable name is typically placed in main, so this defaults to "". |
| xlim | range of the x-axis. NULL (default) uses a pretty range of the non-missing values in x. |
| heights | numeric vector of relative panel heights for layout(): three values for histogram/boxplot/ecdf, two for histogram/boxplot or histogram/ecdf only. NULL (default) chooses automatically. |
| hist | controls the histogram panel. TRUE (default) uses package defaults; FALSE/NA suppresses the panel (xlim then falls back to the pretty range of the data); a list overrides specific arguments forwarded to graphics::hist(). The element type selects the plot style: "hist" (standard histogram, chosen automatically for continuous or high-cardinality data) or "mass" (vertical bars per unique value, for discrete/low-cardinality data). The element panel.last adds to the histogram after it is drawn: a function without arguments or a quoted call, e.g. list(panel.last = quote(abline(v = 0, lty = 2))). |
| dens | controls the kernel density curve. TRUE (default) draws a curve via stats::density(); a list overrides specific arguments (e.g. list(bw = 0.1, col = "red")). |
| rug | controls a rug plot. FALSE (default) suppresses it; TRUE or a list draw a rug via graphics::rug(). |
| curve | controls a fitted theoretical distribution curve on the histogram. FALSE (default), NULL, or NA suppress it; TRUE draws a normal curve with mean(x)/sd(x); a list may include expr as a function of x or a one-sided formula in x for any distribution (e.g. list(expr = ~ dt(x, df = 2), col = "darkgreen")). The formula is evaluated in the environment it was written in, so local variables may be referenced (see the gamma example below). |
| boxplot | controls the boxplot panel. TRUE (default) draws a horizontal boxplot with a mean marker and CI band; FALSE/ NA suppresses the panel. A list overrides arguments forwarded to graphics::boxplot(); two extra elements control the mean display: pch.mean (default 3) and col.meanci (default getTheme()$grid$col). Set either to NA to suppress that element. |
| ecdf | controls the ECDF panel. TRUE (default) calls plotECDF(); a list overrides specific arguments. |
| curveEcdf | controls a fitted theoretical CDF curve on the ECDF panel, analogous to curve. FALSE (default) suppresses it. |

| | |
|---|---|
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to par() via the internal framework. Note that mar given here sets the *outer* margins (oma) of the multi-panel figure; the inner panel margins are managed internally and cannot be overridden. |

## Details

Each plot component is controlled via a single argument accepting bedrock::callIf() semantics:

- TRUE: draw with package defaults
- FALSE, NULL, or NA: suppress entirely
- a named list: draw with the given overrides merged into the defaults

Performance: for very large vectors (n > 1e7) the density curve, ECDF, and semi-transparent boxplot outliers will still take noticeable time. For exploratory work on very large data, consider sampling first: plotFreqDist(x[sample(length(x), 5000)]).

## Value

No return value (invisible NULL), called for its side effect of drawing the plot.

## See Also

hist(), boxplot(), plotECDF(), density(), rug(), layout(), theme

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotLines(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
plotFreqDist(faithful$eruptions)

# custom histogram breaks, density color, boxplot styling
plotFreqDist(faithful$eruptions,
  hist    = list(breaks = 50),
  dens    = list(col = "olivedrab4"),
  boxplot = list(col = "olivedrab2", pch.mean = NA, col.meanci = NA))

# no density, no ecdf, add rug instead
plotFreqDist(faithful$eruptions,
  dens = FALSE, ecdf = FALSE,
  hist = list(xaxt = "s"),
  rug  = TRUE,
  heights = c(3, 2.5),
  main = "Eruption time")

# overlay a normal density curve
x <- rnorm(1000)
plotFreqDist(x, curve = TRUE, boxplot = FALSE, ecdf = FALSE)

# compare with a t-distribution curve
```

```
plotFreqDist(x,
  curve   = list(expr = ~ dt(x, df = 2), col = "darkgreen"),
  boxplot = FALSE, ecdf = FALSE)

# overlay gamma distribution on both histogram and ECDF
ozone <- airquality$Ozone
m <- mean(ozone, na.rm = TRUE)
v <- var(ozone, na.rm = TRUE)
plotFreqDist(ozone,
  hist       = list(breaks = 15),
  curve      = list(expr = ~ dgamma(x, shape = m^2/v, scale = v/m),
                    col = "navajowhite3"),
  curveEcdf  = list(expr = ~ pgamma(x, shape = m^2/v, scale = v/m),
                    col = "navajowhite3"),
  main = "Airquality - Ozone")
```

---

plotFun *Plot Mathematical Functions*

---

## Description

Flexible plotting of mathematical functions in Cartesian, polar, and parametric form.

## Usage

```
plotFun(
  expr,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  from = 0,
  to = 1,
  n = 500,
  args = NULL,
  add = FALSE,
  col = .useTheme,
  lwd = 1,
  lty = 1,
  grid = .useTheme,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| expr | expression defining the function. Either a formula (y ~ f(x)), a polar formula (r ~ f(phi)), or a list of two formulas for parametric plots. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from expr. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | labels for the axes. |
| xlim, ylim | numeric vectors of length 2 defining axis limits. For Cartesian functions, xlim defaults to c(from, to). For polar and parametric plots, limits are derived from the data. If only xlim is supplied for polar/parametric plots, ylim mirrors it (ensures asp=1 renders correctly). |
| from, to | numeric; lower and upper bound of the parameter domain. Default from=0, to=1. |
| n | integer; number of evaluation points. Default 500. |
| args | named list of additional parameters fixed for the function expression (e.g. list(a = 2) for y ~ sin(a*x)). |
| add | logical; if TRUE, adds to an existing plot without redrawing axes or grid. Default FALSE. |
| col | color of the line. .useTheme (default) resolves to getTheme()$twin[1] - the primary accent color, consistent with lines.loess() and plotQQ(). |
| lwd | line width. Default 1. |
| lty | line type. Default 1. |
| grid | controls drawing of the background grid. .useTheme (default) follows the active theme (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to graphics::par() via the internal framework. |

## Details

The function automatically detects the type of input:

- **Cartesian:** y ~ f(x)
- **Polar:** r ~ f(phi) or r ~ f(theta)
- **Parametric:** list(x ~ f(t), y ~ g(t))

Additional parameters fixed for the expression can be passed via args.

## Value

Invisibly returns a list with components x and y (the plotted coordinates after finite filtering).

## See Also

Other plot.distribution: plotProbDist(), shade()

## Examples

```
# Cartesian
plotFun(y ~ x^2)

# Damped oscillation - add second curve to existing plot
plotFun(y ~ 3*exp(-x/5)*sin(4*x), from = 0, to = 10)
plotFun(y ~ 3*exp(-x/5)*sin(6*x), from = 0, to = 10, add = TRUE)

# Cardioid (polar)
plotFun(r ~ 2*(1 + cos(phi)), from = 0, to = 2*pi)

# Heart curve (parametric)
plotFun(
  list(
    x ~ 13*cos(t) - 5*cos(2*t) - 2*cos(3*t) - cos(4*t),
    y ~ 16*sin(t)^3
  ),
  from = 0, to = 2*pi, lwd = 2
)

# Parameter sweep
for (a in 1:3)
  plotFun(y ~ sin(a*x), args = list(a = a),
          from = 0, to = 2*pi,
          add = (a != 1), col = a)
```

---

# plotHeatmap *Heatmap for Categorical Data*

---

## Description

Visualizes a contingency table using a heatmap representation. Cell values are mapped to colors based on counts or proportions, optionally with text labels overlaid.

## Usage

```
plotHeatmap(
  x,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  zlim = NULL,
  scale = c("count", "prop", "row", "col"),
  col = .useTheme,
  border = NA,
```

```
  naCol = "gray90",
  box = .useTheme,
  text = FALSE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a contingency table, matrix, or a pair of categorical vectors coercible via table(). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the expression passed as x (e.g. "tab"). "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | label for the x-axis. |
| ylab | label for the y-axis. |
| xlim, ylim | numeric vectors of length 2 specifying axis limits. |
| zlim | numeric vector of length 2 specifying the range used for color scaling. If NULL, the range of the data is used. |
| scale | character specifying how values are computed:<br>"count" absolute frequencies<br>"prop" joint proportions $P(X, Y)$<br>"row" row-wise proportions $P(Y \mid X)$<br>"col" column-wise proportions $P(X \mid Y)$ |
| col | optional vector of colors. Default is a hardcoded sequential white-to-navy ramp (pal("Blues", n = 100)) - deliberately not theme-driven: cell values here are sequential (one direction, no sign change), unlike the active theme's categorical palette or diverging twin pair, neither of which fits a heat scale. |
| border | color of tile borders. Defaults to NA. |
| naCol | color used for missing values. |
| box | controls drawing of the outer frame around the tile grid, drawn via rect() at the exact cell boundaries rather than graphics::box() (the initial plot suppresses the standard box via frame.plot = FALSE, since cell bounds differ from the default plot region). .useTheme (default) resolves border color/width from getTheme()$box. TRUE/FALSE, or a named list overriding rect() arguments for this call only. |
| text | logical; if TRUE, cell values are printed on top of the tiles using fm() formatting. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, or an explicit string, as for .withGraphicsState() (internal). |
| ... | further graphical parameters passed to graphics::par() via the internal framework. |

## Details

The heatmap represents values in a contingency table using color intensity. Depending on `scale`, the plot shows either absolute counts or different types of proportions. Rows are drawn in reading order: the first table row appears at the top, matching the printed table and the other bivariate plots. This plot complements association and spine plots by focusing on overall structure rather than conditional distributions or statistical inference.

## Value

Invisibly returns the matrix used for plotting.

## See Also

`plotAssoc()`, `graphics::image()`, theme

Other plot.bivariate: `plotAssoc()`, `plotBag()`, `plotBubble()`, `plotCor()`, `plotDens2D()`, `plotHexbin()`, `plotMosaic()`, `plotXY()`

## Examples

```
# applicants by department and gender
tab <- margin.table(UCBAdmissions, c(3, 2))

plotHeatmap(tab,
            scale = "prop",
            text = TRUE)

# share of the genders within the departments
plotHeatmap(tab,
            scale = "row",
            text = TRUE,
            main = "Applicants by department")
```

---

# plotHexbin *Hexagonal Binning Plot*

---

## Description

Displays a two-dimensional density estimate using hexagonal bins. Observations are aggregated into hexagons and coloured according to the number of observations falling into each cell.

## Usage

```
plotHexbin(
  x,
  y,
  main = NULL,
  xlab = "",
```

```
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  nBins = 30,
  col = NULL,
  border = NA,
  grid = .useTheme,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of x-values. |
| y | numeric vector of y-values. |
| main | main title. |
| xlab | label for the x-axis. |
| ylab | label for the y-axis. |
| xlim | limits for the x-axis. |
| ylim | limits for the y-axis. |
| nBins | number of hexagons across the x-axis. |
| col | colours used for the count scale. If NULL, a default sequential palette is used. |
| border | border colour of the hexagons. |
| grid | controls the background grid. .useTheme (default) follows the active theme (getTheme()$grid), TRUE draws it, FALSE, NULL or NA suppress it, and a named list is passed to graphics::grid(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to .applyParFromDots(). |

## Value

Invisibly returns a list containing the computed hexbin object and the original x and y.

## See Also

Other plot.bivariate: plotAssoc(), plotBag(), plotBubble(), plotCor(), plotDens2D(), plotHeatmap(), plotMosaic(), plotXY()

## Examples

```
if (requireNamespace("hexbin", quietly = TRUE)) {

  set.seed(42)
  x <- rnorm(5000)
  y <- x + rnorm(5000)
```

```
  plotHexbin(x, y)

  # fewer and larger cells, user-defined colors
  plotHexbin(x, y, nBins = 15,
             col = hcl.colors(50, "YlOrRd", rev = TRUE), border = "white",
             main = "Hexagonal binning")
}
```

---

`plotLift` *Lift Chart*

---

## Description

Draws the cumulative lift curve, the cumulative gain curve, or the per-group lift bars of a binary classifier, together with the baseline of a random model. The chart answers the operational question directly: how much better is acting on the top-scored share of the cases than acting on a random share of the same size.

## Usage

```
plotLift(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ylim = NULL,
  type = c("cumulative", "gain", "decile"),
  col = .useTheme,
  lwd = 2,
  grid = .useTheme,
  box = .useTheme,
  baseline = TRUE,
  perfect = FALSE,
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `"Lift"`, as returned by `alloy::lift()`. |
| `main` | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then derived from the expression passed as `x`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |

| | |
|---|---|
| xlab | label for the x-axis. NULL (default) derives a label from type. |
| ylab | label for the y-axis. NULL (default) derives a label from type. |
| ylim | numeric vector of length 2; y-axis limits. NULL (default) spans the curve together with the baseline. |
| type | the curve to draw. One of "cumulative" (cumulative lift over depth, the default), "gain" (share of all positives captured, over depth), or "decile" (per-group lift as bars). |
| col | color of the curve or the bars. .useTheme (default) resolves to getTheme()$twin[1] - a single accent color, consistent with plotECDF(). |
| lwd | line width of the curve. Has no effect for type = "decile". |
| grid | controls drawing of the background grid. Can be:<br>• .useTheme (default): follow the active theme (getTheme()$grid)<br>• TRUE: draw grid with theme settings<br>• FALSE, NULL, or NA: suppress grid<br>• a named list: arguments passed to graphics::grid(), overriding the theme defaults for this call only |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for grid. |
| baseline | controls the reference line of a random model - a horizontal line at 1 for type = "cumulative" and "decile", the diagonal for type = "gain". Can be:<br>• TRUE (default): draw with default settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to graphics::lines() (or graphics::abline() for type = "decile"), e.g. list(col = "black", lty = "dotted") |
| perfect | controls the curve of a perfect ranking - the theoretical maximum attainable at each depth, given the base rate. FALSE by default, since it compresses the interesting part of the y-axis; TRUE or a named list to draw it, as for baseline. Has no effect for type = "decile". |
| legend | controls drawing of the legend. Can be:<br>• TRUE (default): draw with default settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to graphics::legend(), e.g. list(x = "bottomleft") |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list of arguments for stamp(). |
| ... | further graphical parameters passed to par() via the internal framework. |

## Details

Reading the cumulative curve at depth 0.2 gives the factor by which contacting the top-scored fifth of the cases beats contacting a random fifth. The gain variant answers the complementary question - what share of all positives that fifth captures.

The curve necessarily converges to 1 (cumulative lift) or to the diagonal endpoint (gain) at depth 1, where all cases are selected and the model no longer discriminates. Only the left part of the curve carries decision value: a model that separates well over the first two deciles and poorly afterwards is preferable for a small campaign to one with the reverse profile and identical AUC.

Optional plot components (`grid`, `box`, `baseline`, `perfect`, `legend`) follow `bedrock::callIf()` semantics:

- `TRUE`: draw with defaults
- `FALSE`, `NULL`, or `NA`: suppress component
- named list: customize component arguments

`col`, `grid`, `box`, and `stamp` default to `.useTheme`, deferring to the package's active theme (see theme) rather than a hardcoded value.

The number of groups is a property of the lift table, not of the plot - set it via the `nBins` argument of `alloy::lift()`.

## Value

Invisibly returns `x`.

## See Also

`alloy::lift()`, `alloy::roc()`, `plotECDF()`, `bedrock::callIf()`, theme

Other plot.special: `plotBinaryTree()`, `plotCirc()`, `plotMiss()`, `plotPolar()`, `plotPropCI()`, `plotSankey()`, `plotTernary()`, `plotTimeSeries()`, `plotTreemap()`, `plotWeb()`

## Examples

```
# A lift table is normally computed by alloy::lift(). To keep the example
# self-contained, the columns used by the plot are built by hand here:
# cases ranked by decreasing score and cut into equally sized groups.
liftTable <- function(score, y, nBins = 10) {
  y    <- y[order(score, decreasing = TRUE)]
  bin  <- cut(seq_along(y), nBins, labels = FALSE)
  nObs <- as.vector(table(bin))
  nPos <- as.vector(tapply(y, bin, sum))
  structure(
    data.frame(bin     = seq_len(nBins),
               lift    = (nPos / nObs) / mean(y),
               depth   = cumsum(nObs) / length(y),
               cumLift = (cumsum(nPos) / cumsum(nObs)) / mean(y),
               gain    = cumsum(nPos) / sum(y)),
    class    = c("Lift", "data.frame"),
    baseRate = mean(y))
}

y   <- as.integer(iris$Species == "virginica")
fit <- glm(y ~ Sepal.Length + Sepal.Width, data = iris, family = binomial)
lft <- liftTable(fitted(fit), y)
```

```
plotLift(lft)
plotLift(lft, type = "gain")

# Per-group bars, coarser grouping set at computation time
plotLift(liftTable(fitted(fit), y, nBins = 5), type = "decile")

# Add the perfect-ranking reference, suppress the legend
plotLift(lft, perfect = TRUE, legend = FALSE)

# No title, compact top margin
plotLift(lft, main = "")
```

---

`plotLines` *Line Plot for Multiple Series*

---

## Description

Draws one or several line series using `graphics::matplot()`. The function accepts either a matrix of values or separate `x` and `y` coordinates and supports optional point symbols, grid lines, and an automatically positioned legend.

## Usage

```
plotLines(
  x,
  y,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  xax = NULL,
  yax = NULL,
  lty = 1,
  lwd = 2,
  col = .useTheme,
  points = FALSE,
  grid = .useTheme,
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | numeric vector, matrix or data frame. If `y` is missing, `x` is interpreted as a matrix of series where rows correspond to x positions and columns to individual lines. |

| | |
|---|---|
| y | optional numeric vector or matrix giving the y-values. If supplied, x is interpreted as the x-coordinates. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | labels for the axes. |
| xlim, ylim | limits for the axes. |
| xax, yax | axis specification passed to graphics::axis(). |
| lty | line type(s). |
| lwd | line width(s). |
| col | colours for the lines. .useTheme (default) resolves to pal(getTheme()$palette), the active theme's qualitative palette. |
| points | controls drawing of points on the lines. FALSE (default) suppresses points; TRUE draws with theme defaults (getTheme()$points); a named list overrides individual elements (pch, col, bg, cex). |
| grid | controls drawing of the background grid. .useTheme (default) follows the active theme (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| legend | controls the legend. TRUE (default) draws an inline legend via textLegend at the last value of each series. FALSE/NA suppresses it. A list overrides legend arguments. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to graphics::par() via .applyParFromDots() and to the plotting functions. |

## Details

If y is missing, x is interpreted as a matrix and each column is drawn as a separate line. Row names are used for the x-axis labels if available. The legend labels default to the column names of the data.

## Value

Invisibly returns a list containing:

- x the x-values used for plotting,
- y the y-values (if supplied),
- legend the legend specification if drawn.

## See Also

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotQQ(), plotRidge(), plotViolin()

## Examples

```
m <- matrix(c(3,4,5,1,5,4,2,6,2), nrow = 3,
            dimnames = list(
              dose = c("A","B","C"),
              age  = c("2000","2001","2002")
            ))

plotLines(m, lwd = 2, main = "Dose ~ Age")

# with points
plotLines(m, points = TRUE)

# custom legend
plotLines(m, legend = list(cex = 0.8))
```

---

`plotMiss` *Plot Missing Data*

---

## Description

Takes a data frame and displays the location of missing data. The missings can be clustered and be displayed together.

## Usage

```
plotMiss(
  x,
  main = NULL,
  cluster = FALSE,
  col = "deeppink4",
  bg = fade("navajowhite3", 0.3),
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | a data.frame to be analysed. |
| `main` | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then derived from the expression passed as `x`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |
| `cluster` | logical, defining if the missings should be clustered. Default is `FALSE`. |
| `col` | the colour of the missings. |
| `bg` | the background colour of the plot. |

| | |
|---|---|
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | the dots are passed to plot(). |

## Details

A graphical display of the position of the missings can be help to detect dependencies or patterns within the missings.

## Value

if cluster is set to TRUE, the new order will be returned invisibly.

## Note

Following an idea of Henk Harmsen henk@carbonmetrics.com

## See Also

hclust(), bedrock::countCompCases()

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap(), plotWeb()

## Examples

```
plotMiss(airquality, main="Missing data (in orignal order)")
plotMiss(airquality, main="Missing data (clustered)", cluster=TRUE)
```

---

# plotMosaic *Mosaic Plot for 2-Way Contingency Tables*

---

## Description

Draws a mosaic plot (spine plot) for a 2-way contingency table, with proportional tile areas, optional cell labels (counts or percentages), and a "Few"-style minimal appearance (white tile separators, muted palette, optional legend).

## Usage

```
plotMosaic(
  x,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  swap = FALSE,
  horiz = FALSE,
  gap = 0.01,
```

```
  col = .useTheme,
  border = "white",
  legend = TRUE,
  labels = c("p", "n", "none"),
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a 2-way contingency table, matrix, or array coercible via `as.table()`. Higher-dimensional arrays must be collapsed first, e.g. via `apply(x, c(1,2), sum)`. |
| main | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then derived from the expression passed as `x`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | character or `NULL`. Axis labels. Default `NULL` (no labels), since the category levels together with `main` and the legend title are usually self-explanatory. |
| swap | logical. If `TRUE`, transpose the two dimensions of `x` before plotting, so the second table dimension determines the x-axis split and the first becomes the fill/legend variable. Default `FALSE`. |
| horiz | logical. If `TRUE`, draw the mosaic horizontally: the first table dimension is shown top-to-bottom on the y-axis instead of left-to-right on the x-axis. Default `FALSE`. |
| gap | numeric. Width of the white separator drawn between adjacent tiles, in plot-region units (`[0, 1]`). Default `0.01`. |
| col | vector of colors for the fill/legend variable (the second table dimension, or first if `swap = TRUE`). `.useTheme` (default) resolves to `pal(getTheme()$palette, n = <number of levels>)` - the active theme's qualitative palette (see theme), sampled or interpolated to match the number of category levels. A diverging or sequential color ramp is deliberately not used here: the fill variable is an unordered categorical variable, and a ramp would visually suggest an ordering between levels that doesn't exist. |
| border | color of the tile borders. Default `"white"`. |
| legend | logical. Draw a legend for the fill variable. Default `TRUE`. |
| labels | character, one of `"p"`, `"n"`, `"none"`. Cell labels showing the proportion of the table total (`"p"`), the absolute frequency (`"n"`), or no labels (`"none"`). Labels are only drawn for tiles large enough to hold them. Default `"p"`. A list sets the details: the type (element `type`, or the one unnamed element), the character expansion factor `cex` (default `0.8`) and the number of decimal digits `digits` for percentages (default 1), e.g. `labels = list("p", cex = 0.7, digits = 0)`. |
| stamp | controls the corner stamp. `.useTheme` (default) resolves to `getTheme()$stamp`. `TRUE`/`FALSE`/ `NULL`, or an explicit string, as for `.withGraphicsState()` (internal). |
| ... | further graphical parameters passed to `par()` via `.applyParFromDots()`, e.g. `mar`, `cex.axis`, `las`. |

## Details

Cells belonging to a row with a row total of zero are drawn as zero-height tiles and receive no label, but do not cause an error or NaN in the geometry.

In both orientations only the first table dimension receives an axis (its split points are constant across the whole plot); the second dimension's split points vary per column/row and are represented via the legend only. No numeric probability axis is drawn. The second dimension is stacked top-down (vertical) resp. left-to-right (horiz = TRUE): its first level sits at the top / left, matching graphics::spineplot() and the legend order.

## Value

Invisibly, the data.frame of tile geometry as produced by .computeMosaicTiles(), with columns x0, x1, y0, y1, n, p, pCond and one column per table dimension. This allows users to add further annotations (e.g. via rect() or text()) on top of the plot.

## See Also

plotCatDist(), plotBar(), getTheme()

Other plot.bivariate: plotAssoc(), plotBag(), plotBubble(), plotCor(), plotDens2D(), plotHeatmap(), plotHexbin(), plotXY()

## Examples

```
tab <- apply(HairEyeColor, c(1, 2), sum)

plotMosaic(tab)
plotMosaic(tab, swap = TRUE)
plotMosaic(tab, horiz = TRUE, main = "Hair ~ Eyecolor")
```

---

# plotPolar *Polar Plot for Radial Data*

---

## Description

Creates a polar coordinate plot for one or multiple radial series.

## Usage

```
plotPolar(
  r,
  theta = NULL,
  main = NULL,
  rlim = NULL,
  type = "p",
  add = FALSE,
  col = NULL,
```

```
  border = NULL,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| r | numeric vector or matrix of radial values. Each row represents one series. |
| theta | optional numeric vector or matrix of angles (in radians). Must match the dimensions of r. If NULL, equally spaced angles are used. |
| main | optional plot title. |
| rlim | numeric limit for radial axis. If NULL, determined automatically. |
| type | character vector specifying plot type for each series:<br>**"p"** points<br>**"l"** polygon (connected and optionally filled)<br>**"h"** radial segments ("histogram"-style) |
| add | logical; if TRUE, adds to an existing plot. |
| col | color for points/lines. |
| border | color for border in case of type polygon. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to base plotting functions. |

## Details

The function supports plotting multiple radial series simultaneously. Each row of r is treated as a separate series.

Angles are interpreted in radians. If theta is not provided, points are distributed evenly over $[0, 2\pi)$.

Graphical parameters such as lwd, lty, pch, cex, fill, and mar can be passed via ....

## Value

Invisibly returns NULL.

## See Also

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotMiss(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap(), plotWeb()

## Examples

```
r <- matrix(runif(20), nrow = 2)
plotPolar(r, type = c("l", "p"), col = c("blue", "red"))

# with custom angles
theta <- seq(0, 2*pi, length.out = 10)
plotPolar(r[1,], theta = theta, type = "h")
```

plotProbDist *Plot Probability Distribution*

## Description

Produces a plot of a probability distribution function with shaded areas. Useful for illustrating critical regions, p-values, or probability masses in statistics teaching materials.

## Usage

```
plotProbDist(
  breaks,
  FUN,
  main = NULL,
  ylab = "density",
  xlim = NULL,
  col = NULL,
  density = 7,
  grid = .useTheme,
  box = .useTheme,
  areaLabels = NULL,
  breakLabels = NULL,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| breaks | numeric vector of break points defining the boundaries between shaded areas. The first and last value define the plot range passed to shade(); interior values are the actual boundaries. |
| FUN | the distribution density function to plot, typically of the form function(x) dnorm(x, mean=0, sd=1). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the expression passed as FUN. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| ylab | label for the y-axis. Default is "density". |
| xlim | numeric vector of length 2. x-axis limits passed to curve(). |
| col | color(s) for the shaded areas. Recycled if shorter than the number of areas. Defaults to the "helsana" palette. |
| density | density of shading lines passed to shade(). Default is 7. |
| grid | controls the background grid. .useTheme (default) follows the active theme, TRUE draws it, FALSE, NULL or NA suppress it. A named list overrides individual arguments passed to grid(). |

box — controls the plot box. .useTheme (default) uses the current theme setting. TRUE/FALSE forces the box on or off. A named list overrides individual arguments passed to box().

areaLabels — controls labels placed in the centre of each shaded area. NULL (default) suppresses labels. TRUE uses LETTERS as default labels. A character vector sets explicit labels. A named list overrides individual arguments passed to boxedText() (e.g. list(cex = 3) or list(x = c(-2, 3), y = 0.1) for manual positioning).

breakLabels — controls labels placed on the x-axis at interior break points via mtext(). NULL (default) suppresses labels. TRUE uses LETTERS as default labels. A character vector sets explicit labels. A named list overrides individual arguments (e.g. list(cex = 1.8, font = 1)).

stamp — controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp().

... — further graphical parameters passed to curve() and .applyParFromDots(), e.g. las, col.axis.

## Value

NULL, invisibly.

## See Also

curve()

Other plot.distribution: plotFun(), shade()

## Examples

```
# Normal distribution with two areas
plotProbDist(breaks = c(-10, -1, 12),
             FUN    = function(x) dnorm(x, mean = 2, sd = 2),
             main   = "Normal-Distribution N(2,2)",
             xlim   = c(-7, 10),
             col    = c("deeppink4", "skyblue3"),
             density = c(20, 7),
             breakLabels = TRUE)

# t-distribution with three areas
plotProbDist(breaks = c(-6, -2.3, 1.5, 6),
             FUN    = function(x) dt(x, df = 8),
             main   = "t-Distribution (df=8)",
             xlim   = c(-4, 4),
             col    = c("deeppink4", "skyblue3", "darkorange2"),
             density = c(20, 7),
             areaLabels = FALSE,
             breakLabels = list(text=c("A", "B")))

# Chi-square distribution
plotProbDist(breaks  = c(0, 15, 35),
             FUN     = function(x) dchisq(x, df = 8),
```

```
main    = expression(chi^2~"-Distribution (df=8)"),
xlim    = c(0, 30),
col     = c("deeppink4", "skyblue3"),
density = c(0, 20),
breakLabels = list(text="B"))
```

---

`plotPropCI` *Plot Proportions with Confidence Intervals*

---

## Description

Displays a horizontal bar with nested confidence interval bands from `min(ciLevels)` to `max(ciLevels)` to visualise the uncertainty of a proportion. Bands are drawn in a semi-transparent grey so that the accumulated overlap creates a natural density gradient - darker in the centre, lighter at the edges.

## Usage

```
plotPropCI(
  x,
  main = NULL,
  xlab = "",
  xlim = c(0, 1),
  ciLevels = seq(0.99, 0.8, by = -0.01),
  col = .useTheme,
  ciCol = addOpacity("grey80", 0.12),
  border = NA,
  grid = .useTheme,
  box = FALSE,
  labels = c("", ""),
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A two-column integer matrix where each row represents a group and the two columns contain counts for the two categories. A numeric vector of length 2 is also accepted and will be coerced to a one-row matrix. |
| `main` | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then derived from the expression passed as `x`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |
| `xlab` | label for the x-axis. Default `""`. |
| `xlim` | numeric vector of length 2 for the x-axis limits. Default `c(0, 1)`. |

| | |
|---|---|
| ciLevels | numeric vector of confidence levels for the nested bands. Default seq(0.99, 0.80, by = -0.01) (20 bands, 99\ 80\ bands share the same translucent color and no border. |
| col | character vector of length 2 specifying fill colours for the stacked bar. .useTheme (default) resolves to getTheme()$twin - the active theme's two-color pair. Note this is purely "first label gets the first color"; unlike plotCor()/plotWeb(), there is no positive/ negative sign convention here, since proportions of two arbitrary categories (e.g. "yes"/"no") have no inherent sign. |
| ciCol | colour for the confidence interval bands. Default is a semi-transparent grey (addOpacity("grey80", 0.12)). Deliberately not theme-driven (like the sequential scales in plotDens2D()/plotHeatmap()): this is a structural mechanism (many overlapping translucent bands building a gradient via overdraw), not a categorical or diverging color choice. |
| border | border colour for the confidence interval bands and the stacked bar. Default NA (no border). |
| grid | controls drawing of the background grid (vertical lines at the proportion ticks only - there is no meaningful horizontal grid for the categorical group axis). .useTheme (default) follows the active theme (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| box | controls drawing of the plot box. Default FALSE (no frame, consistent with this chart's minimal "Few"-style appearance). TRUE/NA, or a named list, as for graphics::box(). |
| labels | character vector of length 2 with labels for the two categories, displayed at the top of the plot. Default c("", ""). |
| legend | controls the legend explaining the CI band range. TRUE (default) draws it. FALSE/NA suppresses it. A named list overrides arguments forwarded to graphics::legend(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further arguments passed to graphics::par() via the internal framework, and to graphics::barplot(). |

## Details

Each row of x is displayed as a horizontal stacked bar showing the proportion of the first category. Confidence intervals are calculated using prop.test() and drawn as nested bands per ciLevels, all in the same semi-transparent colour. The repeated overdraw naturally darkens the centre of the interval where all bands overlap. A vertical segment marks the observed proportion.

## Value

Invisibly returns NULL. Called for its side effect of producing a plot.

## See Also

stats::prop.test(), theme

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotMiss(), plotPolar(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap(), plotWeb()

## Examples

```
m <- matrix(c(22, 111, 33, 120, 80, 100), nrow = 3, byrow = TRUE)
plotPropCI(m, labels = c("yes", "no"), main = "Response by Group")

# single row - vector input is accepted
plotPropCI(m[1, ], labels = c("yes", "no"))
```

---

# plotQQ *QQ-Plot for Any Distribution*

---

## Description

Create a QQ-plot for a variable of any distribution. The assumed underlying distribution can be defined as a function of f(p), including all required parameters. Confidence bands are provided by default.

## Usage

```
plotQQ(
  x,
  qFun = qnorm,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  datax = FALSE,
  add = FALSE,
  cex = .useTheme,
  grid = .useTheme,
  box = .useTheme,
  cband = list(conf.level = 0.95),
  qqline = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | the data sample |
| qFun | the quantile function of the assumed distribution. Can either be given as simple function name or defined as own function using the required arguments. Default is `qnorm()`. See examples. |
| main | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then derived from the distribution, e.g. `"Q-Q-Plot (qnorm)"`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |

| | |
|---|---|
| xlab | the xlab for the plot |
| ylab | the ylab for the plot |
| datax | logical. Should data values be on the x-axis? Default is FALSE. |
| add | logical specifying if the points should be added to an already existing plot; defaults to FALSE. |
| cex | character expansion factor for the points. .useTheme (default) resolves to getTheme()$points$cex. |
| grid | controls drawing of the background grid. .useTheme (default) follows the active theme (getTheme()$grid). TRUE/FALSE/NA, or a named list, as for graphics::grid(). |
| box | controls drawing of the plot box. .useTheme (default) resolves to getTheme()$box. TRUE/FALSE/NA, or a named list, as for graphics::box(). |
| cband | controls the confidence band. FALSE, NULL, or NA suppress it. A named list configures it; its conf.level element (default 0.95) controls the confidence level of the pointwise Kolmogorov-Smirnov-based band (conf.level is consumed before the band is drawn and never forwarded as a graphical parameter). Other list elements (e.g. col, border) configure the band's appearance. |
| qqline | arguments for the qqline. This will be estimated as a line through the 25\ procedure as qqline() does for normal distribution (instead of set it to abline(a = 0, b = 1)). The quantiles can however be overwritten by setting the argument probs to some user defined values. Also the method for calculating the quantiles can be defined (default is 7, see quantile()). The line defaults are set to col = par("fg"), lwd = par("lwd") and lty = par("lty"). No line will be plotted if args.qqline is set to NA. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list of arguments for stamp() (e.g. list(text = "...", las = 2)). |
| ... | the dots are passed to the plot function. |

## Details

The function generates a sequence of points between 0 and 1 and transforms those into quantiles by means of the defined assumed distribution.

## Value

No return value (invisible NULL), called for its side effect of drawing the plot.

## Note

The code is inspired by the tip 10.22 "Creating other Quantile-Quantile plots" from R Cookbook and based on R-Core code from the function qqline. The calculation of confidence bands are rewritten based on an algorithm published in the package BoutrosLab.plotting.general.

Based on code by Ying Wu

## References

Teetor, P. (2011) *R Cookbook*. O'Reilly, pp. 254-255.

## See Also

stats::qqnorm, stats::qqline, stats::qqplot, theme, lines.loess

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotRidge(), plotViolin()

## Examples

```
y <- rexp(100, 1/10)
plotQQ(y, function(p) qexp(p, rate=1/10))

w <- rweibull(100, shape=2)
plotQQ(w, qFun = function(p) qweibull(p, shape=4))

z <- rchisq(100, df=5)
plotQQ(z, function(p) qchisq(p, df=5),
       qqline=list(col=2, probs=c(0.1, 0.6)),
       main=expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
abline(0,1)

# 90% confidence band instead of the 95% default
plotQQ(y, function(p) qexp(p, rate=1/10), cband = list(conf.level = 0.90))

# add 5 random sets
for(i in 1:5){
  z <- rchisq(100, df=5)
  plotQQ(z, function(p) qchisq(p, df=5), add=TRUE, qqline = FALSE,
         col="grey", lty="dotted")
}
```

---

# plotRidge *Ridge Plot (Stacked Density Plot)*

---

## Description

Draws stacked kernel density estimates (ridge plot) for grouped data. Each group is displayed as a density curve shifted along the y-axis.

## Usage

```
plotRidge(x, ...)

## Default S3 method:
plotRidge(
  x,
  ...,
  main = NULL,
  xlab = "",
```

```
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  bw = "nrd0",
  normalize = c("global", "group"),
  add = FALSE,
  scale = 1,
  spacing = 1,
  col = NULL,
  border = NULL,
  lwd = 1,
  lty = 1,
  fill = TRUE,
  grid = NA,
  stamp = .useTheme
)

## S3 method for class 'formula'
plotRidge(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = "",
  ylab = "",
  ...
)
```

## Arguments

| | |
|---|---|
| x | a numeric vector, or a list of numeric vectors representing groups. |
| ... | further numeric vectors (unnamed), each forming a group, or graphical parameters passed to par() (named). Unnamed vectors are only allowed when x is a single vector. |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the names of the data arguments, or the formula. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | axis labels. For the formula method, empty labels default to the response and the grouping variable. |
| xlim, ylim | axis limits. |
| bw | bandwidth for stats::density(). |
| normalize | how the density heights are scaled to scale: "global" (default) divides all densities by the highest peak of all groups, so that heights stay comparable across groups (as in ggridges); "group" divides each density by its own peak, so that every ridge reaches the full height. Use the latter when a single narrow group would otherwise flatten all others. |

| | |
|---|---|
| add | logical; if TRUE, adds to an existing plot. |
| scale | scaling factor for density height. |
| spacing | vertical spacing between ridges. |
| col | fill color(s), recycled over the groups. NULL (default) uses the palette. |
| border | border color(s), recycled over the groups. NULL (default) uses col. |
| lwd | line width(s). |
| lty | line type(s). |
| fill | logical; fill area under densities. |
| grid | logical, NA, or list controlling grid. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp(). |
| formula | a formula of the form y ~ group, or y ~ a:b for the cells of several grouping variables. y ~ a + b is not accepted, see bedrock::resolveFormula(). |
| data | an optional data frame containing the variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot(). |
| na.action | a function specifying how missing values are handled, defaults to na.omit(). |

## Details

Ridge plots are useful for comparing distributions across multiple groups. Each density is normalized and vertically offset, improving readability compared to overlaid density plots.

## Value

Invisibly returns NULL.

## See Also

plotDens()

Other plot.univariate: plotArea(), plotBar(), plotBox(), plotCatDist(), plotDens(), plotDensBox(), plotDot(), plotECDF(), plotFreqDist(), plotLines(), plotQQ(), plotViolin()

## Examples

```
set.seed(1)
df <- data.frame(
  value = c(rnorm(100), rnorm(100, 2), rnorm(100, 4)),
  group = rep(c("A", "B", "C"), each = 100)
)

plotRidge(value ~ group, data = df)

# subset and the cells of two grouping variables
plotRidge(len ~ supp:dose, ToothGrowth, subset = len > 8)
```

```
# the narrow group VC:0.5 flattens all others; scale each ridge to its
# own peak instead
plotRidge(len ~ supp:dose, ToothGrowth, subset = len > 8,
          normalize = "group")
```

---

plotSankey *Sankey Diagram (Flow Diagram)*

---

## Description

Draws a Sankey diagram: nodes arranged in stages (columns) and flows between them as ribbons whose width is proportional to their value, as used for money flows, patient pathways or energy balances. The input is a plain list of flows; stages, node order and colors are derived from it.

## Usage

```
plotSankey(
  x,
  stages = NULL,
  main = NULL,
  order = NULL,
  nodeWidth = 0.08,
  gap = 0.03,
  col = .useTheme,
  border = NA,
  flowCol = "source",
  opacity = 0.4,
  labels = TRUE,
  values = FALSE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | a data frame with one row per flow and three columns: the origin node, the destination node and the value of the flow (finite, non-negative). The column names do not matter, only their order. A node may appear as destination in one stage and as origin in the next. Node names identify nodes globally: a state that recurs at different stages (e.g. "Healthy" before and after treatment) needs distinct names, which `labels` can then display alike. |
| stages | optional character vector of column headings, one per stage, e.g. `c("Sources", "Regimes", "Providers")`. |
| main | main title. `NULL` (default) draws none, unless the theme sets `autoTitle = TRUE` (see theme); the title is then the name of `x`. `""`, `NA` or `FALSE` always suppress it and compact the top margin; any other string is used as given. |

| | |
|---|---|
| order | optional character vector of node names giving their order from top to bottom within each stage. Nodes not listed follow in the order of their first appearance in x, which is the default. |
| nodeWidth | width of the nodes as a fraction of the distance between two stages, between 0 and 1. |
| gap | minimal vertical space between the nodes of a stage, as a fraction of the plot height (non-negative). It applies to the stage that needs the most room; the other stages spread their nodes over the same height, so all stages with more than one node are flush at the top and bottom. A stage with a single node is centred. |
| col | node colors. .useTheme (default) gives each stage a hue from the theme palette and its nodes shades of it. A named vector (c(Bund = "darkgreen", ...)) sets the colors of the named nodes, an unnamed one is recycled over all nodes in stage order. |
| border | color of a line along the left and right edge of the nodes, where the ribbons meet them, e.g. "white" for a fine separation; NA (default) draws none. The line width follows lwd in .... Top and bottom stay open, so the nodes keep exactly the height of their ribbons. |
| flowCol | color of the ribbons: "source" (default) takes the color of the origin node, "target" that of the destination node; any other value is used as a color for all ribbons. |
| opacity | opacity of the ribbons, between 0 and 1. |
| labels | controls the node labels. TRUE (default) draws the node names, left of the first stage, right of all others. FALSE, NULL or NA suppress them. A named character vector replaces the text of the named nodes, e.g. c(Healthy.1 = "Healthy", Healthy.2 = "Healthy"). A named list overrides arguments of graphics::text(), e.g. list(cex = 0.7, col = "grey30"). |
| values | append the node value to the labels: FALSE (default) or TRUE, which formats with fm(); a list of arguments for fm(), e.g. list(digits = 1, bigMark = "'"); or a function that turns the values into strings. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to par() via the internal framework. |

## Details

The stage of a node is the length of the longest directed path that leads to it from a node without incoming flows, so the stages follow from the flows alone. Flows that lead back to an earlier node (cycles) are not supported. Zero flows are dropped, and with them nodes that have no other flows.

A flow may skip stages. It is then led through an invisible pass-through node in each stage in between, placed below the real nodes, so that it does not run through a node it has nothing to do with.

The height of a node is the larger of its total inflow and total outflow; unequal totals are allowed. Within a node, the ribbons are stacked in the order of the nodes they lead to or come from, which avoids crossings at the node itself. The order of the nodes within a stage is not optimised; order sets it by hand.

The left and right margins are widened for the labels of the first and last stage, unless `mar` in `...` sets those sides.

## Value

Invisibly, a list with the data frames `nodes` (name, stage, inflow, outflow, value = the larger of both, color and the coordinates of each node) and `flows` (the positive flows with their vertical positions at both ends), e.g. for checking balances or adding annotations.

## See Also

Other plot.special: `plotBinaryTree()`, `plotCirc()`, `plotLift()`, `plotMiss()`, `plotPolar()`, `plotPropCI()`, `plotTernary()`, `plotTimeSeries()`, `plotTreemap()`, `plotWeb()`

## Examples

```
# Financing of the Swiss health care system (simplified, CHF bn):
# one row per flow - origin, destination, value
flows <- data.frame(
  from = c("Confederation", "Confederation", "Cantons", "Cantons",
           "Municipalities", "Companies", "Companies",
           "Households", "Households", "Households",
           "State", "State", "Health insurance", "Health insurance",
           "Social insurance", "Social insurance",
           "Private insurance", "Private insurance",
           "Out-of-pocket", "Out-of-pocket"),
  to   = c("State", "Health insurance", "State", "Health insurance",
           "State", "Health insurance", "Social insurance",
           "Health insurance", "Private insurance", "Out-of-pocket",
           "Hospitals", "Nursing homes", "Hospitals", "Physicians",
           "Hospitals", "Nursing homes", "Hospitals", "Physicians",
           "Physicians", "Dentists"),
  value = c(5, 2, 9, 2, 4, 7, 2, 18, 8, 12,
            12, 6, 14, 15, 2, 3, 4, 4, 7, 5)
)

plotSankey(flows, stages = c("Sources", "Financing", "Providers"))

# own colors for some nodes, ribbons colored by destination, values
plotSankey(flows, col = c(Households = "darkgreen", State = "orange"),
           flowCol = "target", values = TRUE, border = "white", lwd = 2)

# a flow that skips a stage is led around the nodes in between
plotSankey(data.frame(from = c("A", "B", "A"), to = c("B", "C", "C"),
                      value = c(5, 5, 3)))

# the same state at two stages: distinct names, same label
path <- data.frame(from = c("Healthy.1", "Healthy.1", "Ill", "Ill"),
                   to   = c("Healthy.2", "Ill", "Healthy.2", "Dead"),
                   value = c(70, 30, 20, 10))
plotSankey(path, labels = c(Healthy.1 = "Healthy", Healthy.2 = "Healthy"))
```

plotTernary *Ternary Plot*

## Description

Draws a ternary plot for compositional data consisting of three non-negative components per observation. Each row is interpreted as a composition and internally rescaled to sum to 1 if necessary.

## Usage

```
plotTernary(
  x,
  y = NULL,
  z = NULL,
  main = "",
  xlim = c(-1, 1),
  ylim = c(-0.5, 1),
  add = FALSE,
  col = NULL,
  pch = 16,
  cex = 1,
  grid = NA,
  labels = NULL,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A numeric vector, matrix, or data frame. If y and z are provided, x, y, and z are combined into a matrix. Otherwise, x must contain exactly three columns. |
| y | optional numeric vector for the second component. |
| z | optional numeric vector for the third component. |
| main | plot title. |
| xlim, ylim | plot limits (usually left at defaults). |
| add | logical; if TRUE, adds to an existing plot. |
| col | point color(s). |
| pch | plotting symbol. |
| cex | point size. |
| grid | logical, NA, or list controlling the ternary grid. |
| labels | character vector of length 3 specifying axis labels. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to par(). |

## Details

A ternary plot represents three-part compositions on an equilateral triangle. Each observation $(x_1, x_2, x_3)$ is mapped to 2D coordinates using barycentric transformation. If rows do not sum to 1, they are automatically normalized with a warning.

Graphical elements such as grids are controlled via the unified plot design system using `bedrock::callIf()` and `.theme()`.

## Value

Invisibly returns `NULL`.

## See Also

`plotDens()`, `plotRidge()`

Other plot.special: `plotBinaryTree()`, `plotCirc()`, `plotLift()`, `plotMiss()`, `plotPolar()`, `plotPropCI()`, `plotSankey()`, `plotTimeSeries()`, `plotTreemap()`, `plotWeb()`

## Examples

```
set.seed(1)
x <- runif(100)
y <- runif(100)
z <- runif(100)

plotTernary(x, y, z)

# matrix input
M <- cbind(x, y, z)
plotTernary(M, labels = c("A", "B", "C"))
```

---

# plotTimeSeries *Combined Plot of a Time Series and Its ACF and PACF*

---

## Description

Combined plot of a time Series and its autocorrelation and partial autocorrelation

## Usage

```
plotTimeSeries(
  x,
  main = NULL,
  ylab = NULL,
  maxLag = 10 * log10(length(x)),
  stamp = .useTheme,
  ...
)
```

### Arguments

| | |
|---|---|
| x | univariate time series. |
| main | an overall title for the plot |
| ylab | a title for the y axis: see title(). |
| maxLag | integer. Defines the number of lags to be displayed. The default is 10 * log10(length(series)). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | the dots are passed to the plot command. |

### Details

plotTimeSeries plots a combination of the time series and its autocorrelation and partial autocorrelation.

### Value

x, invisibly. Called for its side effect of drawing the plot.

### Note

Rewritten based on ideas of M.Huerzeler

### See Also

ts()

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotMiss(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTreemap(), plotWeb()

### Examples

```
plotTimeSeries(AirPassengers)
```

---

## plotTreemap *Treemap Plot*

---

### Description

Draws a treemap in which the area of each rectangle is proportional to the corresponding value in x. Optionally, rectangles can be grouped into higher-level regions.

## Usage

```
plotTreemap(
  x,
  groups = NULL,
  main = NULL,
  area = NULL,
  labels = NULL,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of positive values determining the rectangle sizes. |
| groups | optional grouping variable. Values sharing the same group are placed within a common enclosing region. A list sets the appearance of the group regions as well: it holds the grouping variable (element by, or the one unnamed element) and the elements area and labels, which follow the conventions of the arguments area and labels, e.g. groups = list(grp, area = list(border = "black", lwd = 2)). By default, group names are used as labels when more than one group is present. |
| main | main title of the plot. |
| area | controls the appearance of individual rectangles.<br>• NULL or TRUE: use defaults.<br>• FALSE or NA: suppress rectangle fill.<br>• Atomic vector: interpreted as col.<br>• List: graphical parameters such as col, border, and lwd. |
| labels | controls the labels of individual rectangles.<br>• NULL or TRUE: use default labels (names(x)).<br>• FALSE or NA: suppress labels.<br>• Character vector: interpreted as label text.<br>• List: label properties such as text, col, and cex. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | additional graphical parameters passed to .applyParFromDots(). |

## Details

The appearance of individual rectangles is controlled through the area and labels arguments, that of the groups through the elements area and labels of a groups list. These accept logical values, vectors, or lists.

Individual rectangles are sized according to the values in x. When groups is supplied, a treemap is first constructed for the groups, and each group's area is then subdivided among its members.

The arguments area and labels - and their counterparts in a groups list - provide a flexible interface for controlling the appearance of the plot while keeping the main function signature compact.

## Value

Invisibly returns a list containing the coordinates of group centres and the centres of their child rectangles.

## See Also

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotMiss(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotWeb()

## Examples

```
x <- c(A = 6, B = 5, C = 4, D = 3, E = 2, F = 1)

plotTreemap(x)

plotTreemap(
  x,
  labels = list(col = "white")
)

grp <- c("G1", "G1", "G1", "G2", "G2", "G2")

plotTreemap(
  x,
  groups = list(grp, labels = TRUE)
)

plotTreemap(
  x,
  groups = list(grp, area = list(border = "black", lwd = 2)),
  area = terrain.colors(length(x)),
  labels = FALSE
)
```

---

# plotViolin *Violin Plot*

---

## Description

Draws violin plots for one or more groups, combining kernel density estimation with optional boxplot overlays. The function follows a boxplot-like interface and supports both default and formula methods.

## Usage

```
plotViolin(x, ...)

## Default S3 method:
```

```
plotViolin(
  x,
  ...,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  bw = "nrd0",
  trim = TRUE,
  horiz = FALSE,
  at = NULL,
  names = NULL,
  add = FALSE,
  col = "grey80",
  border = "black",
  lwd = 1,
  grid = NA,
  box = TRUE,
  quantiles = NULL,
  stamp = .useTheme
)

## S3 method for class 'formula'
plotViolin(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = "",
  ylab = "",
  horiz = FALSE,
  names = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector, list of numeric vectors, or first group. |
| ... | additional data vectors (unnamed) or graphical parameters passed to par(). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: the names of the data arguments, or the formula. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab, ylab | axis labels. For the formula method, empty labels default to the grouping variable and the response. |

| | |
|---|---|
| xlim, ylim | axis limits. NULL (default) uses the range of the densities, padded by 2%; given limits are used as they are. |
| bw | bandwidth specification passed to density(). |
| trim | logical. If TRUE (default), the kernel density estimate of each group is restricted to the observed data range (from = min(x), to = max(x)), so the violin never extends beyond the actual data — matching the default behavior of ggplot2::geom_violin(). If FALSE, density() is called with its own defaults, which extend the tails up to cut * bw beyond range(x) and may produce violins that reach into implausible values (e.g. scores above 100 or below 0). |
| horiz | logical; if TRUE, draws horizontal violins. |
| at | numeric positions of the groups. |
| names | optional group labels. |
| add | logical; if TRUE, adds to an existing plot. |
| col | fill color(s) of the violins. |
| border | border color(s) of the violins. |
| lwd | line width for violin borders. |
| grid | logical, NA, or list controlling background grid. |
| box | logical or list controlling the boxplot overlay (see Details). |
| quantiles | optional numeric vector of probabilities for drawing quantile lines inside each violin. |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/NULL, a string, or a named list for stamp(). |
| formula | a formula of the form y ~ group, or y ~ a:b for the cells of several grouping variables. y ~ a + b is not accepted (unlike boxplot()), see bedrock::resolveFormula(). |
| data | an optional data frame containing the variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = len > 10), as in boxplot(). |
| na.action | a function specifying how missing values are handled, defaults to na.omit(). |

## Details

The violin shape is constructed from a kernel density estimate of each group, scaled to a fixed maximum width. Optionally, boxplots and quantile lines can be added.

Graphical elements such as the boxplot overlay and grid are controlled via a flexible interface using TRUE, FALSE, NA, or list(...) and are evaluated using bedrock::callIf().

## Value

Invisibly returns NULL.

### Data Handling

The function accepts:

- a numeric vector
- multiple vectors via ...
- a list of numeric vectors

Groups are handled similarly to `boxplot()`.

### See Also

`boxplot()`, `density()`

Other plot.univariate: `plotArea()`, `plotBar()`, `plotBox()`, `plotCatDist()`, `plotDens()`, `plotDensBox()`, `plotDot()`, `plotECDF()`, `plotFreqDist()`, `plotLines()`, `plotQQ()`, `plotRidge()`

### Examples

```
set.seed(1)
x <- rnorm(100)
y <- rnorm(100, 1)

plotViolin(x, y)

# horizontal violins
plotViolin(x, y, horiz = TRUE)

# with quantiles
plotViolin(x, y, quantiles = c(0.25, 0.5, 0.75))

# untrimmed: tails extend beyond the observed data range
plotViolin(x, y, trim = FALSE)

# custom styling
plotViolin(x, y,
  col = c("lightblue", "salmon"),
  box = list(col = "white"),
  grid = TRUE
)

# formula interface
df <- data.frame(
  value = rnorm(200),
  group = rep(letters[1:4], each = 50)
)

plotViolin(value ~ group, data = df)

# subset and the cells of two grouping variables
plotViolin(len ~ supp:dose, ToothGrowth, subset = len > 8)
```

plotWeb *Plot a Web of Connected Points*

## Description

Displays a symmetric matrix (typically a correlation matrix) as a network diagram: variables are placed equidistantly around a circle, and connecting lines between each pair are drawn with width proportional to the absolute value of the matrix entry and color indicating its sign.

## Usage

```
plotWeb(
  m,
  main = NULL,
  dist = 0.5,
  col = hcol("red", "blue"),
  lty = NULL,
  lwd = NULL,
  labels = TRUE,
  points = list(pch = 21, cex = 2, col = "black", bg = "darkgrey"),
  legend = TRUE,
  stamp = .useTheme,
  ...
)
```

## Arguments

| | |
|---|---|
| `m` | a symmetric numeric matrix (e.g. a correlation matrix). |
| `main` | main title of the plot. `NULL` (default) produces no title. |
| `dist` | distance of node labels from the outer circle. Default `0.5`. |
| `col` | two colors for the connecting lines: the first is used for negative values, the second for positive values. `.useTheme` (default) resolves to `getTheme()$twin` - consistent with the sign-based coloring in `plotCor()`. |
| `lty` | line type for the connecting lines. `NULL` (default) inherits from `par("lty")`. |
| `lwd` | line widths for the connecting lines. `NULL` (default) scales widths linearly between 0.5 and 10 in proportion to the absolute matrix values. |
| `labels` | controls node labels around the circle. `TRUE` (default) draws labels using `colnames(m)` with default styling. `FALSE/NA/NULL` suppresses labels. A character vector is taken as the label texts. A list holds the texts and the details: |
| | `text` character vector of label texts (or the one unnamed element of the list); defaults to `colnames(m)` |
| | `las` orientation: `1` horizontal (default), `2` radial, `3` vertical |
| | `adj` label adjustment (0/0.5/1); `NULL` (default) chooses automatically based on position around the circle |

| | |
|---|---|
| | cex character expansion; default 1.0 |
| points | controls the node point symbols. The default (list(pch=21, cex=2, col="black", bg="darkgrey")) draws prominent filled circles, larger than the generic data-point default, since nodes represent variables rather than observations. FALSE/NA/NULL suppresses the points. A named list overrides individual elements (pch, cex, col, bg). |
| legend | controls the legend. TRUE (default) draws a legend showing the line widths and colors for the minimum/maximum positive and negative values. FALSE/NA suppresses it. A named list overrides arguments forwarded to graphics::legend(). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| ... | further graphical parameters passed to graphics::par() and to the internal canvas() call. |

## Details

The function uses the lower triangular matrix of m; when overriding lwd or col manually, values must be supplied in the same order as m[lower.tri(m)].

## Value

Invisibly returns a list of x/y coordinates of the node points, useful for adding further annotations to the plot.

## See Also

plotCor(), theme

Other plot.special: plotBinaryTree(), plotCirc(), plotLift(), plotMiss(), plotPolar(), plotPropCI(), plotSankey(), plotTernary(), plotTimeSeries(), plotTreemap()

## Examples

```
m <- cor(swiss)
plotWeb(m, main = "Swiss correlation")

# custom labels (abbreviations)
plotWeb(m, labels = list(abbreviate(colnames(m), 4), cex = 0.9))

# show only significant correlations
p <- outer(
  (vars <- colnames(mtcars)), vars,
  Vectorize(function(v1, v2)
    cor.test(mtcars[[v1]], mtcars[[v2]])$p.value)
)
dimnames(p) <- list(vars, vars)
m2 <- cor(mtcars)
m2[p > 0.05] <- NA
plotWeb(m2, labels = list(las = 2))
```

plotXY *Scatterplot with Optional Smooth Lines*

## Description

Draws a scatterplot of two numeric variables with optional linear and locally weighted regression lines, and an optional legend.

## Usage

```
plotXY(x, ...)

## Default S3 method:
plotXY(
  x,
  y,
  main = NULL,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  col = .useTheme,
  bg = .useTheme,
  pch = .useTheme,
  cex = .useTheme,
  grid = .useTheme,
  box = .useTheme,
  lm = TRUE,
  loess = TRUE,
  legend = TRUE,
  stamp = .useTheme,
  ...
)

## S3 method for class 'formula'
plotXY(
  formula,
  data,
  subset,
  na.action = na.omit,
  main = NULL,
  xlab = "",
  ylab = "",
  ...
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of x-values, or a formula of the form y ~ x. |
| ... | further graphical parameters passed to par() via the internal framework. |
| y | numeric vector of y-values (ignored if a formula is used). |
| main | main title. NULL (default) draws none, unless the theme sets autoTitle = TRUE (see theme); the title is then derived from the input: y ~ x for the default method, or the formula. "", NA or FALSE always suppress it and compact the top margin; any other string is used as given. |
| xlab | label for the x-axis. |
| ylab | label for the y-axis. |
| xlim | numeric vector of length 2; x-axis limits. If NULL (default), the range of x is used. |
| ylim | numeric vector of length 2; y-axis limits. If NULL (default), the range of y is used. |
| col | color of the points. .useTheme (default) resolves to getTheme()$points$col. |
| bg | background (fill) color of the points. .useTheme (default) resolves to getTheme()$points$bg. |
| pch | plotting character. .useTheme (default) resolves to getTheme()$points$pch. |
| cex | character expansion factor for points. .useTheme (default) resolves to getTheme()$points$cex. |
| grid | controls drawing of the background grid. Can be:<br>• .useTheme (default): follow the active theme (getTheme()$grid)<br>• TRUE: draw grid with theme settings<br>• FALSE, NULL, or NA: suppress grid<br>• a named list: arguments passed to graphics::grid(), overriding the theme defaults for this call only |
| box | controls drawing of the plot box. Can be:<br>• .useTheme (default): follow the active theme (getTheme()$box)<br>• TRUE: draw box with theme settings<br>• FALSE, NULL, or NA: suppress box<br>• a named list: arguments passed to graphics::box(), overriding the theme defaults for this call only |
| lm | controls drawing of the linear regression line. Can be:<br>• TRUE: draw with default settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to graphics::lines(), e.g. list(col = "blue", lwd = 2) |
| loess | controls drawing of the locally weighted regression line. Can be:<br>• TRUE: draw with default settings<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to graphics::lines(), e.g. list(col = "red", lty = "dashed") |

| | |
|---|---|
| legend | controls drawing of the legend. Can be:<br>• TRUE: draw with default settings (position "topright")<br>• FALSE, NULL, or NA: suppress<br>• a named list: arguments passed to graphics::legend(), e.g. list(x = "bottomleft")<br>The legend is only drawn when at least one of lm or loess is active. lm/loess line colors are taken from the active theme's twin colors (getTheme()$twin). |
| stamp | controls the corner stamp. .useTheme (default) resolves to getTheme()$stamp. TRUE/FALSE/ NULL, a string, or a named list for stamp(). |
| formula | a formula of the form y ~ x, both numeric. |
| data | an optional data frame containing variables in the formula. |
| subset | an optional expression indicating which observations to use, evaluated in data (subset = delivery_min < 40), as in plot.formula(). |
| na.action | a function specifying how missing values are handled. Defaults to na.omit. |

## Details

Optional plot components (grid, box, lm, loess, legend) follow bedrock::callIf() semantics:

- TRUE: draw with defaults
- FALSE, NULL, or NA: suppress component
- named list: customize component arguments

col, bg, pch, cex, grid, and box default to .useTheme, deferring to the package's active theme (see theme) rather than a hardcoded value. This means setTheme(list(points = list(col = "black"))) changes the point color for every call to plotXY() (and any other function using the same theme section) that doesn't override col explicitly.

## Value

Invisibly returns NULL.

## See Also

graphics::plot(), stats::lm(), stats::loess(), bedrock::callIf()

Other plot.bivariate: plotAssoc(), plotBag(), plotBubble(), plotCor(), plotDens2D(), plotHeatmap(), plotHexbin(), plotMosaic()

## Examples

```
plotXY(temperature ~ delivery_min, bedrock::Pizza,
       main = "Temperature vs. Delivery Time")

# Suppress loess, customize lm line
plotXY(temperature ~ delivery_min, bedrock::Pizza,
       lm    = list(col = "darkred", lwd = 2),
       loess = FALSE)
```

```
# No title, compact top margin
plotXY(temperature ~ delivery_min, bedrock::Pizza, main = "")

# subset, evaluated in data
plotXY(temperature ~ delivery_min, bedrock::Pizza,
       subset = area == "Camden")
```

---

`polarGrid` *Draw a Polar Grid with Optional Labels*

---

## Description

Adds a polar coordinate grid (circles and radial lines) to an existing plot. Optionally includes labels for radii and angles.

## Usage

```
polarGrid(
  nRadial = NULL,
  nAngular = NULL,
  radialLabels = NULL,
  angleLabels = NULL,
  labelRadians = FALSE,
  col = "lightgray",
  lty = "dotted",
  lwd = par("lwd"),
  cex = 1,
  las = 1,
  adj = NULL,
  dist = NULL
)
```

## Arguments

| | |
|---|---|
| `nRadial` | numeric or vector controlling radial grid lines:<br>`NULL` Uses default "pretty" axis values.<br>**single numeric** Number of radial grid lines.<br>**numeric vector** Explicit radii at which to draw circles.<br>**all** `NA` Suppress radial grid lines. |
| `nAngular` | numeric or vector controlling angular grid lines:<br>`NULL` Uses 12 equally spaced angles.<br>**single numeric** Number of angular divisions.<br>**numeric vector** Explicit angles (in radians).<br>**all** `NA` Suppress angular grid lines. |

| | |
|---|---|
| radialLabels | optional labels for radial grid lines (excluding zero). If NULL, labels are generated automatically. Use NA to suppress labels. |
| angleLabels | optional labels for angular grid lines. If NULL, labels are generated automatically (degrees or radians). Use NA to suppress labels. |
| labelRadians | logical; if TRUE, angle labels are shown in radians, otherwise in degrees. |
| col | color of grid lines. |
| lty | line type for grid lines. |
| lwd | line width for grid lines. |
| cex | character expansion factor for the labels. |
| las | integer controlling label orientation (as in graphics::par()). |
| adj | numeric vector specifying text justification. |
| dist | numeric distance from origin for angular labels. |

## Details

This function is intended to be used together with polar plotting functions such as plotPolar. It assumes an existing plot with equal aspect ratio.

Radial grid lines are drawn as concentric circles, while angular grid lines are drawn as segments from the origin.

Label placement and formatting can be customized via adj, las, and dist.

## Value

Invisibly returns NULL.

## See Also

grid()

Other graphics.setup: canvas(), setBackCol()

## Examples

```
plot(0, 0, type = "n", xlim = c(-1, 1), ylim = c(-1, 1), asp = 1)
polarGrid()

# custom grid
plot(0, 0, type = "n", xlim = c(-2, 2), ylim = c(-2, 2), asp = 1)
polarGrid(nRadial = 4, nAngular = 8, col = "gray")

# suppress labels
polarGrid(radialLabels = NA, angleLabels = NA)
```

`polygonX` *Draw Polygonal Geometries*

## Description

Generic function for drawing polygon-based geometry objects. This function extends graphics::polygon() with support for geometry objects such as circle(), ellipse(), regPolygon() and ring() and further remains fully compatible with its original interface. #' For ordinary coordinate vectors the call is forwarded to `graphics::polygon()`. Geometry objects such as `circle()`, `ellipse()`, `regPolygon()` and `ring()` are dispatched to specialised methods.

## Usage

```
polygonX(x, ...)

## Default S3 method:
polygonX(
  x,
  y = NULL,
  density = NULL,
  angle = 45,
  border = NULL,
  col = NA,
  lty = par("lty"),
  ...,
  fillOddEven = FALSE
)

## S3 method for class 'ringGeometry'
polygonX(x, rule = "evenodd", ...)

## S3 method for class 'polygonGeometry'
polygonX(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object to be drawn. |
| `...` | further arguments passed to the corresponding method. |
| `y` | numeric vector of y-coordinates. |
| `density` | density of shading lines. |
| `angle` | angle of shading lines in degrees. |
| `border` | border colour. |
| `col` | fill colour. |
| `lty` | line type. |
| `fillOddEven` | logical; should the odd-even rule be used for filling? |

`rule` character string specifying the filling rule passed to `graphics::polypath()`. One of `"evenodd"` or `"winding"`.

## Value

Invisibly returns `x`.

## See Also

Other geometry.structures: `arc()`, `band()`, `bezier()`, `circle()`, `ellipse()`, `regPolygon()`, `ring()`

## Examples

```
canvas()

polygonX(
  circle(radius = 1),
  col = "lightblue"
)

polygonX(
  regPolygon(
    radius = 0.7,
    nVertices = 5
  ),
  border = "red"
)
```

---

# Prefix *Metric Prefixes*

---

## Description

SI metric prefixes from yotta ($10^{24}$) down to yocto ($10^{-24}$), with their name, symbol and multiplier. Used as default lookup table by `convUnit` and for SI-prefix formatting in `fm`.

## Usage

```
Prefix
```

## Format

A `data.frame` with 20 rows and 3 columns:

**pref** `character`, name of the prefix (e.g. `"kilo"`).

**abbr** `character`, symbol of the prefix (e.g. `"k"`); `"u"` is used for micro.

**mult** `numeric`, multiplier, i.e. the power of ten the prefix stands for (e.g. `1000`).

## Value

A `data.frame` as described under *Format*; the object is a constant, not a function.

## Examples

```
Prefix
Prefix[Prefix$abbr == "k", ]
```

---

`preview` *Preview an Object*

---

## Description

Generic function for an explicit, on-demand preview of an object, as distinct from `print()`. `preview()` exists for object types where `print()` is shared with another package's S3 generic dispatch (e.g. class `"html"`, used both by **pharos** and **htmltools** for genuinely different purposes), so that registering an own `print.*` method would silently overwrite - or be overwritten by - the other package's behaviour.

## Usage

```
preview(x, ...)

## Default S3 method:
preview(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | object to preview. |
| `...` | further arguments passed to methods. |

## Details

The default method simply calls `print()`, so `preview()` is always safe to call even for types with no dedicated method.

## Value

`x`, invisibly. Called for its side effect of displaying the object.

## See Also

`asHtml()`

Other html: `asFileLink()`, `asHtml()`, `asImg()`, `embedFile()`, `escapeHtml()`, `htmlNotation`, `htmlSubscript`, `preview.html()`, `toHtmlTable()`

## Examples

```
# without a dedicated method the object is simply printed
preview(1:3)

# HTML markup is rendered as text
preview(asHtml("<b>Note:</b> R<sup>2</sup> = 0.87"))
```

---

`preview.html` *Print HTML markup as readable text*

---

## Description

Renders an `"html"` object as text in the console: tags are stripped or translated (`<sub>`/`<sup>` become _/^, `<b>`/`<strong>`/`<i>`/`<em>` become bold/italic via ANSI codes), common HTML entities (` `, `β`, ...) are decoded, and `<table>` blocks are rendered as aligned text tables.

## Usage

```
## S3 method for class 'html'
preview(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `"html"` |
| `...` | further arguments, currently unused (kept for consistency with `print()`) |

## Details

If output does not support ANSI styling, bold/italic markup is rendered as plain text (handled automatically by **cli**).

## Value

`x`, invisibly

## See Also

Other html: `asFileLink()`, `asHtml()`, `asImg()`, `embedFile()`, `escapeHtml()`, `htmlNotation`, `htmlSubscript`, `preview()`, `toHtmlTable()`

## Examples

```
preview(asHtml(
  "<b>Model:</b> y = &beta;<sub>0</sub> + &beta;<sub>1</sub>x + &varepsilon;"
))

# tables are rendered as aligned text
m <- matrix(c(12.3, 4.56, 7.8, 91.2), nrow = 2,
            dimnames = list(c("a", "b"), c("mean", "sd")))
preview(toHtmlTable(m))
```

---

`print.Unit` *Print Object with Unit*

---

## Description

S3 method for printing objects with a `"Unit"` class. Displays the value along with its associated unit.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | an object with class `"Unit"`. |
| `...` | additional arguments passed to `print()`. |

## Value

Invisibly returns `x`.

## See Also

base::attr, bedrock::label

Other format: `convUnit()`, `fm()`, `fmCI()`, `style()`, `unit()`

## Examples

```
x <- 10
unit(x) <- "m"
class(x) <- "Unit"
print(x)
```

regPolygon *Regular Polygon Geometry*

## Description

Create a regular polygon.

## Usage

```
regPolygon(x = 0, y = 0, radius = 1, nVertices = 6, startAngle = 0)
```

## Arguments

| | |
|---|---|
| x, y | centre coordinates. |
| radius | circumradius. |
| nVertices | number of vertices. |
| startAngle | staring angle in radians. |

## Value

An object inheriting from class "regPolygonGeometry".

## See Also

Other geometry.structures: arc(), band(), bezier(), circle(), ellipse(), polygonX(), ring()

## Examples

```
canvas(xlim = c(-2, 2))

# a hexagon is the default
polygonX(regPolygon(radius = 1.8), col = "lightblue")

# a square, once standing on a vertex and once on an edge
polygonX(regPolygon(radius = 1.2, nVertices = 4), border = "red")
polygonX(regPolygon(radius = 1.2, nVertices = 4, startAngle = pi/4))

# a triangle pointing upwards
polygonX(regPolygon(radius = 0.5, nVertices = 3, startAngle = pi/2),
         col = "royalblue")
```

## rgbToCmy *Convert RGB to CMY*

### Description

Convert RGB colors to the CMY color space.

### Usage

```
rgbToCmy(col, maxColorValue = 1)
```

### Arguments

| | |
|---|---|
| col | RGB matrix with one color per row (columns red, green and blue), or a vector of hexadecimal colors. |
| maxColorValue | maximum channel value of col. Use 255 for hexadecimal colors. |

### Value

Numeric CMY matrix with one color per row.

### See Also

color-conversion-overview

### Examples

```
# one color per row, channels on a scale from 0 to 1
rgbToCmy(matrix(c(1, 0.39, 0.28), nrow = 1))

# the channels of hexadecimal colors run from 0 to 255
rgbToCmy(c("#FF6347", "#4169E1"), maxColorValue = 255)
```

## rgbToCol *Convert RGB Colors to the Nearest Named R Color*

### Description

Finds, for each color, the perceptually closest of the named R colors (grDevices::colors()).

### Usage

```
rgbToCol(col, method = c("ciede2000", "cie76"))
```

### Arguments

| | |
|---|---|
| `col` | colors: an RGB matrix with one color per column and channels on a scale from 0 to 255, as returned by `colToRgb()`, or a vector of R colors (hexadecimal or named). An alpha channel is ignored. |
| `method` | color difference formula, `"ciede2000"` (default) or `"cie76"`, see Details. |

### Details

The colors are converted from sRGB to the CIELAB color space with `grDevices::convertColor()` (reference white D65), where distances approximate perceived differences far better than in RGB or HSV. Two color difference formulas are available:

`"ciede2000"` CIEDE2000, the current CIE recommendation. It corrects the shortcomings of CIE76 in the blue region and for saturated and near-neutral colors.

`"cie76"` CIE76, the Euclidean distance in CIELAB. Simpler, slightly less accurate.

### Value

Character vector of named R colors, one per color in `col`; `NA` for a missing color.

### References

Sharma, G., Wu, W., & Dalal, E. N. (2005). The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1), 21–30. doi:10.1002/col.20070

### See Also

color-conversion-overview

### Examples

```
rgbToCol(matrix(c(162, 42, 42), nrow = 3))
rgbToCol(c("#A22A2A", "#4169E0"))

# CIE76 instead of CIEDE2000
rgbToCol(c("#A22A2A", "#4169E0"), method = "cie76")
```

---

## rgbToHex *Convert RGB to Hexadecimal Colors*

---

### Description

Convert RGB values to hexadecimal color strings.

## Usage

```
rgbToHex(col)
```

## Arguments

| | |
|---|---|
| col | RGB matrix with one color per column, as returned by colToRgb(), or a vector with the channel values of a single color. |

## Value

Character vector of hexadecimal colors, one per color.

## See Also

color-conversion-overview

## Examples

```
rgbToHex(c(162, 42, 42))

# one color per column, as returned by colToRgb()
rgbToHex(colToRgb(c("tomato", "royalblue")))
```

---

# rgbToLong *Convert RGB to Long Integers*

---

## Description

Encode RGB colors as long integers.

## Usage

```
rgbToLong(col)
```

## Arguments

| | |
|---|---|
| col | RGB matrix. |

## Value

Integer vector.

## See Also

color-conversion-overview

## Examples

```
rgbToLong(colToRgb(c("red", "green", "blue")))
```

---

`ring` *Ring Geometry*

---

## Description

Create one or more rings or ring segments.

## Usage

```
ring(
  x = 0,
  y = 0,
  innerRadius = 0.5,
  outerRadius = 1,
  startAngle = 0,
  endAngle = 2 * pi,
  nPoints = 100
)
```

## Arguments

| | |
|---|---|
| `x`, `y` | centre coordinates. |
| `innerRadius` | radius of the inner boundary. |
| `outerRadius` | radius of the outer boundary. |
| `startAngle`, `endAngle` | start and end angle in radians. |
| `nPoints` | number of points used for each boundary. |

## Value

An object inheriting from class `"ringGeometry"` or a `"geometryCollection"`.

## See Also

Other geometry.structures: `arc()`, `band()`, `bezier()`, `circle()`, `ellipse()`, `polygonX()`, `regPolygon()`

## Examples

```
canvas(xlim = c(-2, 2))

# a full ring
polygonX(ring(innerRadius = 1.4, outerRadius = 1.8), col = "lightblue")

# two ring segments at once
r <- ring(innerRadius = c(0.3, 0.8), outerRadius = c(0.7, 1.2),
          startAngle = c(0, pi), endAngle = c(pi/2, 3*pi/2))
polygonX(r, col = c("tomato", "royalblue"))
```

---

# rotate *Rotate a Geometric Structure*

---

## Description

Rotate a geometric structure by an angle theta around a centerpoint xy.

## Usage

```
rotate(x, y = NULL, mx = NULL, my = NULL, theta = pi/3, asp = 1)
```

## Arguments

| | |
|---|---|
| x, y | vectors containing the coordinates of the vertices of the polygon , which has to be rotated. The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, .... See `xy.coords()`. |
| mx, my | xy-coordinates of the center of the rotation. If left to NULL, the centroid of the structure will be used. |
| theta | angle of the rotation |
| asp | the aspect ratio for the rotation. Helpful for rotate structures along an ellipse. |

## Value

The function invisibly returns a list of the coordinates for the rotated shape(s).

## See Also

`polygonX()`, `regPolygon()`, `ellipse()`, `arc()`

Other geometry.transformation: `transformXY()`

## Examples

```
op <- par(no.readonly = TRUE)
# let's have a triangle
canvas(main="Rotation")
x <- regPolygon(nVertices=3)

# and rotate
sapply( (0:3) * pi/6, function(theta) {
  xy <- rotate( x=x, theta=theta )
  polygon(xy, col=addOpacity("blue", 0.2))
} )

abline(v=0, h=0)

par(op)
```

---

# setBackCol *Background of a Plot*

---

## Description

Paints the background of the plot, using either the figure region, the plot region or both. It can sometimes be cumbersome to elaborate the coordinates and base R does not provide a simple function for that.

## Usage

```
setBackCol(col = "grey", region = c("plot", "figure"), border = NA)
```

## Arguments

| | |
|---|---|
| col | the color of the background, if two colors are provided, the first is used for the plot region and the second for the figure region. |
| region | either "plot" or "figure" |
| border | color for rectangle border(s). Default is NA for no borders. |

## Value

No return value (invisible NULL), called for its side effect of painting the background of the current plot.

## See Also

rect()

Other graphics.setup: canvas(), polarGrid()

### Examples

```
# use two different colors for the figure region and the plot region
plot(x = rnorm(100), col="blue", cex=1.2, pch=16,
     panel.first={setBackCol(c("red", "lightyellow"))
                  grid()})
```

---

shade *Produce a shaded Curve*

---

### Description

Sometimes the area under a density curve has to be color shaded, for instance to illustrate a p-value or a specific region under the normal curve. This function draws a curve corresponding to a function over the interval `[from, to]`. It can plot also an expression in the variable `xname`, default `x`.

### Usage

```
shade(expr, col = par("fg"), breaks, density = 10, n = 101, xname = "x", ...)
```

### Arguments

| | |
|---|---|
| `expr` | the name of a function, or a `call()` or an `expression()` written as a function of `x` which will evaluate to an object of the same length as `x`. |
| `col` | color to fill or shade the shape with. The default is taken from `par("fg")`. |
| `breaks` | numeric, a vector giving the breakpoints between the distinct areas to be shaded differently. Should be finite as there are no plots with infinite limits. |
| `density` | the density of the lines as needed in polygon. |
| `n` | integer; the number of x values at which to evaluate. Default is 101. |
| `xname` | character string giving the name to be used for the x axis. |
| `...` | the dots are passed on to `graphics::polygon()`. |

### Details

Useful for shading the area under a curve as often needed for explaining significance tests.

### Value

A list with components `x` and `y` of the points that were drawn is returned invisibly.

### See Also

`graphics::polygon()`, `curve()`

Other plot.distribution: `plotFun()`, `plotProbDist()`

## Examples

```
curve(dt(x, df=5), xlim=c(-6,6),
    main=paste("Student t-Distribution Probability Density Function, df = ", 5, ")", sep=""),
      type="n", las=1, ylab="probability", xlab="t")

shade(dt(x, df=5), breaks=c(-6, qt(0.025, df=5), qt(0.975, df=5), 6),
      col=c("deeppink4", "skyblue3"), density=c(20, 7))
```

---

`splineCI` *Add a Spline Smoother*

---

## Description

Fit a smoothing spline and optionally add confidence bands.

## Usage

```
splineX(x, ...)

## Default S3 method:
splineX(x, ...)

## S3 method for class 'formula'
splineX(formula, data, subset, na.action = na.omit, weights, ...)

## S3 method for class 'SplineX'
lines(
  x,
  col = pal()[1],
  lwd = 2,
  lty = "solid",
  type = "l",
  band = list(conf.level = 0.95),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | spline object returned by `splineX()`. |
| `...` | for `splineX()`, further arguments passed to `stats::smooth.spline()`; for `lines()`, further graphical parameters passed to `lines()`. |
| `formula` | A formula of the form `lhs ~ rhs`, where `lhs` gives the response values and `rhs` the corresponding groups or explanatory variables. |

| | |
|---|---|
| data | an optional matrix or data frame (or similar; see stats::model.frame()) containing the variables in the formula. By default the variables are taken from environment(formula). |
| subset | an optional vector specifying a subset of observations to be used in the analysis. |
| na.action | A function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action"). |
| weights | optional vector of weights, a column of data or a vector of the length of the original data. It is subject to subset and na.action like the variables of the formula. |
| col | line color of the smoother. |
| lwd | line width. |
| lty | line type. |
| type | plotting type passed to lines(). |
| band | controls the confidence band. May be TRUE, FALSE, NULL, NA, or a named list. The confidence level is specified via conf.level, all other elements are graphical parameters of the band. Default is list(conf.level = 0.95). |

## Details

Confidence bands are controlled via band. These arguments can be:

- FALSE, NULL or NA: suppress the band
- TRUE: draw the band with default settings
- a named list: customize the band appearance and confidence level

## Value

splineX() returns the fit of stats::smooth.spline() with the additional class "SplineX". The lines() method returns no value (invisible NULL); it is called for its side effect of adding the smoother and its confidence band to the current plot.

## See Also

loess(), scatter.smooth()

Other graphics.trendlines: lines.lm(), lines.loess()

## Examples

```
op <- par(no.readonly = TRUE)
par(mfrow = c(1, 2))

x <- runif(100)
y <- rnorm(100)

plot(x, y)
lines(splineX(y ~ x))
```

```
plot(dist ~ speed, cars)
lines(splineX(dist ~ speed, cars))

# subset and weights, both evaluated in data
plot(dist ~ speed, cars)
lines(splineX(dist ~ speed, cars, subset = speed > 10,
              weights = sqrt(speed)))

plot(dist ~ speed, cars)
lines(
  splineX(dist ~ speed, cars),
  band = list(
    conf.level = 0.99,
    col = addOpacity("red", 0.3),
    border = "black"
  )
)

par(op)
```

---

`spreadOut` *Spread Out a Vector of Numbers To a Minimum Interval*

---

## Description

Spread the numbers of a vector so that there is a minimum interval between any two numbers (in ascending or descending order). This is helpful when we want to place textboxes on a plot and ensure, that they do not mutually overlap.

## Usage

```
spreadOut(x, minDist = NULL, cex = 1)
```

## Arguments

| | |
|---|---|
| `x` | a numeric vector which may contain `NA`s. |
| `minDist` | the minimum interval between any two values. If this is left to `NULL` (default) the function will check if a plot is open and then use 90%% of `strheight()`. |
| `cex` | numeric character expansion factor; multiplied by `[par]("cex")` yields the final character size; the default `NULL` is equivalent to `1`. |

## Details

`spreadOut()` starts at or near the middle of the vector and increases the intervals between the ordered values. `NA`s are preserved. `spreadOut()` first tries to spread groups of values with intervals less than `minDist` out neatly away from the mean of the group. If this doesn't entirely succeed, a second pass that forces values away from the middle is performed.

`spreadOut()` can also be used to avoid overplotting of axis tick labels where they may be close together.

## Value

On success, the spread out values. If there are less than two valid values, the original vector is returned.

## Note

This function is based on `plotrix::spreadOut()` and has been integrated here with some minor changes.

Based on code by Jim Lemon jim@bitwrit.com.au

## See Also

`strheight()`

Other graphics.layout: `abcCoords()`, `axTicksPosixct()`, `axisBreak()`, `isValidPlotRegion()`, `lineToUser()`, `mar()`, `plotFacet()`

## Examples

```
spreadOut(c(1, 3, 3, 3, 3, 5), 0.2)
spreadOut(c(1, 2.5, 2.5, 3.5, 3.5, 5), 0.2)
spreadOut(c(5, 2.5, 2.5, NA, 3.5, 1, 3.5, NA), 0.2)

# this will almost always invoke the brute force second pass
spreadOut(rnorm(10), 0.5)
```

---

# stamp *Date/Time/Directory Stamp the Current Plot*

---

## Description

Stamp the current plot in the extreme lower right corner. A free text or expression can be defined as text to the stamp.

## Usage

```
stamp(text = .useTheme, las = NULL, cex = 0.6, col = "grey40")
```

## Arguments

| | |
|---|---|
| `text` | character string, expression, or toggle controlling the stamp text. `.useTheme` (default) or `TRUE` resolve to `getTheme()$stamp`, evaluated lazily at draw time. `FALSE`, `NULL`, or `NA` suppress the stamp. Any other string or unevaluated `expression()` is used as given. |
| `las` | orientation; see `graphics::par()`. `NULL` (default) inherits the current `par("las")`. `las = 3` places the stamp vertically along the right edge instead of horizontally along the bottom. |
| `cex, col` | size and color of the stamp text. |

## Details

The default text is taken from the active theme (see theme). If user and date should be included by default, an unevaluated expression, which is evaluated when the stamp is drawn, will help:

```
setTheme(list(stamp = expression(gettextf("%s / %s",
  Sys.getenv("USERNAME"), fm(Sys.Date(), fmt = "yyyy-MM-dd")))))
```

For R results may not be satisfactory if `par(mfrow=)` is in effect.

## Value

The text of the stamp as a character string, invisibly; an empty string if the stamp is suppressed.

## See Also

`text()`

Other graphics.annotation: `barText()`, `boxedText()`, `colLegend()`, `errBars()`, `textLegend()`, `titleRect()`

## Examples

```
plot(1:20)
stamp()
```

---

# strAbbr *Abbreviate Strings Uniquely*

---

## Description

Creates abbreviations of character strings such that they remain distinguishable within the input vector.

## Usage

```
strAbbr(x, minChar = 1, method = c("left", "fix"))
```

## Arguments

| | |
|---|---|
| `x` | a character vector |
| `minChar` | integer; minimum number of characters to retain |
| `method` | character string specifying the abbreviation strategy:<br>• `"left"`: abbreviate each string individually to the shortest unique prefix<br>• `"fix"`: use a common prefix length for all strings such that they are distinguishable |

### Details

The function ensures that abbreviations are unique (within the given vector) while respecting the minimum length `minChar`.

For `method = "left"`, each string is shortened individually to the shortest prefix that distinguishes it from all others.

For `method = "fix"`, a single prefix length is chosen such that all strings are distinguishable.

Unicode-aware substring operations are performed using the **stringi** package.

### Value

A character vector of abbreviated strings.

### See Also

`stringi::stri_sub()`, `stringi::stri_length()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- c("apple", "apricot", "banana")

# individual minimal abbreviations
strAbbr(x, method = "left")

# fixed length abbreviations
strAbbr(x, method = "fix")
```

---

## strAlign *Align Strings*

---

### Description

Aligns a character vector to the left, right, center, or with respect to the first occurrence of a specified separator.

### Usage

```
strAlign(x, sep = "\\r")
```

### Arguments

| | |
|---|---|
| `x` | a character vector |
| `sep` | a character specifying the alignment mode:<br>• `"\\l"`: left alignment<br>• `"\\r"`: right alignment (default)<br>• `"\\c"`: centered alignment<br>• any other character: align at the first occurrence of this separator |

## Details

Alignment is achieved by padding strings with spaces so that all elements have equal width. This is mainly useful for monospaced output.

For left, right, and center alignment, strings are padded to the maximum width using spaces.

If a separator is provided, strings are aligned such that the first occurrence of the separator is vertically aligned. If a string does not contain the separator, it is treated as if the separator occurred at the end.

This function uses Unicode-aware string handling via the **stringi** package.

## Value

A character vector of aligned strings.

## See Also

`stringi::stri_pad()`, `stringi::stri_sub()`, `stringi::stri_trim_right()`

string-overview for an overview of all string utilities in pharos.

## Examples

```
x <- c("here", "there", "everywhere")

# right align (default)
strAlign(x)

# left align
strAlign(x, "\\l")

# center align
strAlign(x, "\\c")

# align at decimal separator
z <- c("6.0", "45.12", "784")
strAlign(z, ".")
```

---

# strCap *Capitalize Strings*

---

## Description

Capitalizes character strings in different ways: first letter, each word, or title case.

## Usage

```
strCap(x, method = c("first", "word", "title"))
```

### Arguments

| | |
|---|---|
| x | a character vector |
| method | character string specifying the capitalization method:<br>• "first": capitalize the first letter of the string<br>• "word": capitalize the first letter of each word<br>• "title": title case, excluding common stopwords |

### Details

The function uses Unicode-aware transformations from the **stringi** package.

For method = "title", common stopwords (e.g., "a", "the", "of") remain lowercase unless they appear as part of another word.

### Value

A character vector with capitalized strings.

### See Also

stringi::stri_trans_totitle(), stringi::stri_split_boundaries()

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- c("hello world", "the lord of the rings")

# first letter
strCap(x, "first")

# each word
strCap(x, "word")

# title case
strCap(x, "title")
```

---

## strChop *Split a String into a Number of Sections of Defined Length*

---

### Description

Splitting a string into a number of sections of defined length is needed, when we want to split a table given as a number of lines without separator into columns. The cutting points can either be defined by the lengths of the sections or directly by position.

## Usage

```
strChop(x, len = NULL, pos = NULL)
```

## Arguments

| | |
|---|---|
| x | the string to be cut in pieces |
| len | a vector with the lengths of the pieces |
| pos | a vector of cutting positions. Will be ignored when len has been defined. |

## Details

If length is going over the end of the string the last part will be returned, so if the rest of the string is needed, it's possible to simply enter a big number as last partlength.

len and pos can't be defined simultaneously, only alternatively.

Typical usages are

```
strChop(x, len) strChop(x, pos)
```

## Value

a vector with the parts of the string.

## See Also

strLeft(), substr()

string-overview for an overview of all string utilities in pharos.

## Examples

```
x <- paste(letters, collapse="")
strChop(x=x, len = c(3,5,2))

# and with the rest integrated
strChop(x=x, len = c(3, 5, 2, nchar(x)))

# cutpoints at 5th and 10th position
strChop(x=x, pos=c(5, 10))
```

## strCountWords *Count Words in Strings*

### Description

Counts the number of words in each element of a character vector.

### Usage

```
strCountWords(x)
```

### Arguments

| | |
|---|---|
| x | a character vector |

### Details

Words are detected using Unicode-aware word boundaries as implemented in **stringi**. This ensures robust handling of different languages, punctuation, and whitespace.

### Value

An integer vector giving the number of words in each element of x.

### See Also

stringi::stri_count_words()

string-overview for an overview of all string utilities in pharos.

### Examples

```
strCountWords("This is a sentence.")

strCountWords(c("One word", "Two words here", NA))
```

## strDist *Compute Distances Between Strings*

### Description

strDist computes distances between strings following to Levenshtein or Hamming method.

## Usage

```
strDist(
  x,
  y,
  method = c("levenshtein", "normlevenshtein", "hamming"),
  mismatch = 1,
  gap = 1,
  ignoreCase = FALSE
)
```

## Arguments

| | |
|---|---|
| x | character vector, first string |
| y | character vector, second string |
| method | character, name of the distance method. This must be "levenshtein", "normlevenshtein" or "hamming". Default is "levenshtein", the classical Levenshtein distance. |
| mismatch | numeric, distance value for a mismatch between symbols |
| gap | numeric, distance value for inserting a gap |
| ignoreCase | if FALSE (default), the distance measure will be case sensitive and if TRUE, case is ignored |

## Details

The function computes the Hamming and the Levenshtein (edit) distance of two given strings (sequences). The Hamming distance between two vectors is the number of mismatches between corresponding entries.

In case of the Hamming distance the two strings must have the same length.

In case of the Levenshtein (edit) distance a scoring and a trace-back matrix are computed and are saved as attributes "ScoringMatrix" and "TraceBackMatrix". The numbers in the trace-back matrix reflect insertion of a gap in string y (1), match/mismatch (2), and insertion of a gap in string x (3).

The edit distance is useful, but normalizing the distance to fall within the interval [0,1] is preferred because it is somewhat difficult to judge whether an LD of for example 4 suggests a high or low degree of similarity. The method "normlevenshtein" for normalizing the LD is sensitive to this scenario. In this implementation, the Levenshtein distance is transformed to fall in this interval as follows:

$$lnd = 1 - \frac{ld}{max(length(x), length(y))}$$

where ld is the edit distance and max(length(x), length(y)) denotes that we divide by the length of the larger of the two character strings. This normalization, referred to as the Levenshtein normalized distance (lnd), yields a statistic where 1 indicates perfect agreement between the two strings, and a 0 denotes imperfect agreement. The closer a value is to 1, the more certain we can be that the character strings are the same; the closer to 0, the less certain.

## Value

strDist returns an object of class "dist"; cf. stats::dist().

### Note

For distances between strings and for string alignments see also Bioconductor package **Biostrings**

Based on code by Matthias Kohl, adapted to conform to package standards.

### References

R. Merkl and S. Waack (2009) *Bioinformatik Interaktiv*. Wiley.

Harold C. Doran (2010) *MiscPsycho. An R Package for Miscellaneous Psychometric Analyses*

### See Also

`utils::adist()`, `stats::dist()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- "GACGGATTATG"
y <- "GATCGGAATAG"
## Levenshtein distance
d <- strDist(x, y)
d
attr(d, "ScoringMatrix")
attr(d, "TraceBackMatrix")

## Hamming distance
strDist(x, y, method="hamming")
```

---

## strExtract *Extract First Match from Strings*

---

### Description

Extracts the first substring matching a regular expression from each element of a character vector.

### Usage

```
strExtract(x, pattern, ...)
```

### Arguments

| | |
|---|---|
| `x` | a character vector |
| `pattern` | a character string containing a regular expression to match |
| `...` | additional arguments passed to `stringi::stri_extract_first_regex()` |

### Details

This function is a thin wrapper around `stringi::stri_extract_first_regex()` providing a simplified interface for extracting the first match of a pattern.

### Value

A character vector containing the first match for each element of `x`. If no match is found, `NA` is returned.

### See Also

`stringi::stri_extract_first_regex()`, `strExtractBetween()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- c("abc123", "no digits", "456xyz")

# extract first number
strExtract(x, "\\d+")

# extract words
strExtract("hello world", "\\w+")
```

---

## strExtractBetween *Extract Substrings Between Patterns*

---

### Description

Extracts substrings that occur between two delimiters.

### Usage

```
strExtractBetween(x, left, right, greedy = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | a character vector |
| `left` | a character string or regular expression marking the left boundary |
| `right` | a character string or regular expression marking the right boundary |
| `greedy` | logical; if `TRUE`, the match is greedy (longest match). If `FALSE` (default), the match is non-greedy (shortest match). |

## Details

The function uses regular expressions to extract the first substring between `left` and `right`. Internally, it constructs a pattern of the form:

- greedy: `left (.*) right`
- non-greedy: `left (.*?) right`

Extraction is performed using `stringi::stri_match_first_regex()`, which returns the first captured group.

## Value

A character vector containing the extracted substrings. If no match is found, `NA` is returned for that element.

## See Also

`stringi::stri_match_first_regex()`, `strExtract()`

string-overview for an overview of all string utilities in pharos.

## Examples

```
x <- c("a[123]b", "x[abc]y", "no match")

# non-greedy (default)
strExtractBetween(x, "\\[", "\\]")

# greedy
strExtractBetween("a[1]b[2]c", "\\[", "\\]", greedy = TRUE)
```

---

`string-overview` *String Functions in pharos*

---

## Description

pharos provides a family of functions for manipulating and inspecting character strings, built on top of **stringi** for Unicode-aware behavior. They fall into two groups:

**Manipulation** – transform, extract, or reshape string content:

| Function | Purpose |
|---|---|
| `strAbbr()` | Abbreviate strings uniquely |
| `strAlign()` | Align strings |
| `strCap()` | Capitalize strings |
| `strChop()` | Split a string into a number of sections of defined length |
| `strExtract()` | Extract first match from strings |
| `strExtractBetween()` | Extract substrings between patterns |

| | |
|---|---|
| strLeft() / strRight() | Return the left or the right part of a string |
| strPad() | Pad a string with justification |
| strRev() | Reverse strings |
| strSpell() | Spell strings using phonetic alphabets |
| strSplit() | Split strings |
| strTrim() | Remove leading/trailing whitespace from a string |
| strTrunc() | Truncate strings and add ellipses if a string is truncated |
| strVal() | Extract numeric values from strings |

**Information** – inspect properties of strings without changing them:

| Function | Purpose |
|---|---|
| strCountWords() | Count words in strings |
| strDist() | Compute distances between strings |
| strIsNumeric() | Check if character strings represent numeric values |
| strLen() | String length |
| strPos() | Find position of first occurrence of a string |

## See Also

base::substr()

---

`strIsNumeric` *Check if Character Strings Represent Numeric Values*

---

## Description

Determines whether elements of a character vector represent valid numeric values. Unlike `as.numeric()`, this function performs a regex-based validation and does not produce warnings.

## Usage

```
strIsNumeric(x, scientific = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a character vector to be tested |
| `scientific` | logical; if `TRUE`, scientific notation (e.g., `"1e3"`) is allowed. Defaults to `FALSE`. |

## Details

The function uses regular expressions via **stringi** to validate numeric formats. Valid formats include optional leading sign (`+` or `-`), optional decimal point, and digits. If `scientific = TRUE`, exponential notation using `e` or `E` is also supported.

Note that special values such as `"Inf"`, `"-Inf"`, and `"NaN"` are not considered numeric by this function.

### Value

A logical vector of the same length as x, indicating whether each element represents a numeric value.

### See Also

as.numeric()

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- c("123", "-3.141", "1e3", "foo", "Inf")

strIsNumeric(x)
strIsNumeric(x, scientific = TRUE)
```

---

## strLeftRight *Returns the Left Or the Right Part Of a String*

---

### Description

Returns the left part or the right part of a string. The number of characters are defined by the argument n. If n is negative, this number of characters will be cut off from the other side.

### Usage

```
strLeft(x, n)

strRight(x, n)
```

### Arguments

| | |
|---|---|
| x | a vector of strings |
| n | a positive or a negative integer, the number of characters to cut. If n is negative, this number of characters will be cut off from the right with strLeft and from the left with strRight.<br>n will be recycled. |

### Details

The functions strLeft and strRight are simple wrappers to substr.

### Value

the left (right) n characters of x

## See Also

string-overview for an overview of all string utilities in pharos.

## Examples

```
strLeft("Hello world!", n=5)
strLeft("Hello world!", n=-5)

strRight("Hello world!", n=6)
strRight("Hello world!", n=-6)

strLeft(c("Lorem", "ipsum", "dolor","sit","amet"), n=2)

strRight(c("Lorem", "ipsum", "dolor","sit","amet"), n=c(2,3))
```

---

# strLen *String length*

---

## Description

Intuitive alias for `nchar()`.

## Usage

```
strLen(x, ...)
```

## Arguments

| | |
|---|---|
| x | a character vector |
| ... | further arguments passed to nchar |

## Value

An integer vector with the number of characters of the elements of x, see `nchar()`.

## See Also

`base::nchar()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
strLen(c("pharos", "", NA))
strLen(month.name)

# a number is converted to a string first
strLen(123456)
```

---

## strPad *Pad a String With Justification*

---

### Description

strPad will fill a string x with defined characters to fit a given length.

### Usage

```
strPad(x, width = NULL, pad = " ", align = c("left", "right", "center"))
```

### Arguments

| | |
|---|---|
| x | a vector of strings to be padded |
| width | resulting width of padded string. If x is a vector and width is left to NULL, it will be set to the length of the largest string in x. |
| pad | string to pad with. Will be repeated as often as necessary. Default is " ". |
| align | adjustment of the old string, one of "left", "right", "center". If set to "left" the old string will be adjusted on the left and the new characters will be filled in on the right side. |

### Details

If a string x has more characters than width, it will be chopped on the length of width.

### Value

the string

### See Also

strAlign(), strTrunc()

string-overview for an overview of all string utilities in pharos.

### Examples

```
strPad("My string", 25, "XoX", "center")
 # [1] "XoXXoXXoMy stringXXoXXoXX"
```

strPos *Find Position of First Occurrence Of a String*

## Description

Returns the numeric position of the first occurrence of a substring within a string. If the search string is not found, the result will be NA.

## Usage

```
strPos(x, pattern, pos = 1)
```

## Arguments

| | |
|---|---|
| x | a character vector in which to search for the pattern, or an object which can be coerced by as.character to a character vector |
| pattern | character string (search string) containing the pattern to be matched in the given character vector. This can be a character string or a regular expression. |
| pos | integer, defining the start position for the search within x. The result will then be relative to the begin of the truncated string. Will be recycled. |

## Details

This is just a wrapper for the function regexpr().

## Value

a vector of the first position of pattern in x

## See Also

strChop(), regexpr()

string-overview for an overview of all string utilities in pharos.

## Examples

```
strPos(x=rownames(mtcars), pattern="y")

# first t, starting with position 4
strPos(x=rownames(mtcars), pattern="t", pos=4)
```

strRev *Reverse Strings*

## Description

Reverses each element of a character vector.

## Usage

```
strRev(x)
```

## Arguments

| | |
|---|---|
| x | a character vector |

## Details

This function uses `stringi::stri_reverse()`, which correctly handles Unicode characters and multi-byte encodings.

## Value

A character vector with each string reversed.

## See Also

`stringi::stri_reverse()`

string-overview for an overview of all string utilities in pharos.

## Examples

```
strRev("abc")

strRev(c("hello", "world"))

# Unicode-safe
strRev("äöü")
```

strSpell *Spell Strings Using Phonetic Alphabets*

## Description

Converts characters in a string into their corresponding phonetic representations using either the NATO phonetic alphabet or Morse code.

## Usage

```
strSpell(x, upr = "CAP", type = c("nato", "morse"))
```

## Arguments

| | |
|---|---|
| x | a character vector (typically of length 1) |
| upr | character string used as a prefix for uppercase letters. Default is "CAP". If NA, no prefix is added. |
| type | character string specifying the encoding system:<br>• "nato": NATO phonetic alphabet (default)<br>• "morse": Morse code |

## Details

Letters (A–Z, a–z) and digits (0–9) are mapped to their corresponding phonetic representations. Other characters are returned unchanged.

For type = "nato", uppercase letters can optionally be prefixed (e.g., "CAP Alfa") to distinguish them from lowercase letters.

The function uses Unicode-aware character splitting via stringi::stri_split_boundaries().

## Value

A character vector with each character of x replaced by its phonetic representation.

## See Also

strTrim(), stringi::stri_split_boundaries()

string-overview for an overview of all string utilities in pharos.

## Examples

```
# NATO spelling
strSpell("ABC")

# with digits
strSpell("A1B2")
```

```
# Morse code
strSpell("SOS", type = "morse")

# without uppercase prefix
strSpell("ABC", upr = NA)
```

---

`strSplit` *Split Strings*

---

## Description

Splits character vectors into substrings. This is a convenience wrapper around `stringi::stri_split_fixed()` and `stringi::stri_split_regex()` with simplified defaults.

## Usage

```
strSplit(x, split = "", fixed = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | a character vector to be split |
| `split` | a character string specifying the delimiter (if `fixed = TRUE`) or a regular expression (if `fixed = FALSE`) |
| `fixed` | logical; if `TRUE`, `split` is treated as a fixed string. Otherwise, it is interpreted as a regular expression. |

## Details

If the input `x` has length 1, the result is returned as a character vector instead of a list for convenience.

This function provides a simplified interface to string splitting using the **stringi** package. It avoids some of the complexity of `base::strsplit()` while providing consistent and Unicode-aware behavior.

## Value

A list of character vectors. If `x` has length 1, a character vector is returned.

## See Also

`stringi::stri_split_fixed()`, `stringi::stri_split_regex()`, `base::strsplit()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
strSplit("a,b,c", ",", fixed = TRUE)

strSplit(c("a b", "c d"), " ")

# regex splitting
strSplit("a1b2c3", "\\d")
```

---

`strTrim` *Remove Leading/Trailing Whitespace From A String*

---

### Description

The function removes whitespace characters as spaces, tabs and newlines from the beginning and end of the supplied string. Whitespace characters occurring in the middle of the string are retained. Trimming with method `"left"` deletes only leading whitespaces, `"right"` only trailing. Designed for users who were socialized by SQL.

### Usage

```
strTrim(x, pattern = " \t\n", method = c("both", "left", "right"))
```

### Arguments

| | |
|---|---|
| `x` | the string to be trimmed |
| `pattern` | the pattern of the whitespaces to be deleted, defaults to space, tab and newline: `" \t\n"` |
| `method` | one out of `"both"` (default), `"left"`, `"right"`. Determines on which side the string should be trimmed. |

### Details

The functions are defined depending on method as
`both`: `gsub( pattern=gettextf("^[\%s]+|[\%s]+$", pattern, pattern), replacement="", x=x)`
`left`: `gsub( pattern=gettextf("^[\%s]+",pattern), replacement="", x=x)`
`right`: `gsub( pattern=gettextf("[\%s]+$",pattern), replacement="", x=x)`

### Value

the string x without whitespaces

### See Also

`base::trimws()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
strTrim("  Hello world! ")

strTrim("  Hello world! ", method="left")
strTrim("  Hello world! ", method="right")

# user defined pattern
strTrim(" ..Hello ... world! ", pattern=" \\.")
```

---

## strTrunc *Truncate Strings and Add Ellipses If a String is Truncated.*

---

### Description

Truncates one or more strings to a specified length, adding an ellipsis (...) to those strings that have been truncated. The truncation can also be performed using word boundaries. Use `strAlign()` to justify the strings if needed.

### Usage

```
strTrunc(x, maxLen = 20, ellipsis = "...", wordBound = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | a vector of strings |
| `maxLen` | the maximum length of the returned strings (NOT counting the appended ellipsis). `maxLen` is recycled. |
| `ellipsis` | the string to be appended, if the string is longer than the given maximal length. The default is `"..."`. |
| `wordBound` | logical. Determines if the maximal length should be reduced to the next smaller word boundary and so words are not chopped. Default is `FALSE`. |

### Value

The string(s) passed as '`x`' now with a maximum length of '`maxLen`' + 3 (for the ellipsis).

### See Also

`strAlign()`, `strPad()`

string-overview for an overview of all string utilities in pharos.

## Examples

```
x <- c("this is short", "and this is a longer text",
       "whereas this is a much longer story, which could not be told shorter")

# simple truncation on 10 characters
strTrunc(x, maxLen=10)

# NAs remain NA
strTrunc(c(x, NA_character_), maxLen=15, wordBound=TRUE)

# using word boundaries
for(i in 0:20)
  print(strTrunc(x, maxLen=i, wordBound=TRUE))

# compare
for(i in 0:20)
  print(strTrunc(x, maxLen=i, wordBound=FALSE))
```

---

# strVal *Extract Numeric Values from Strings*

---

## Description

Extracts numeric values from character strings using regular expressions. The function supports integers, decimal numbers, and scientific notation.

## Usage

```
strVal(
  x,
  collapse = FALSE,
  output = c("character", "numeric"),
  dec = getOption("OutDec")
)
```

## Arguments

| | |
|---|---|
| x | a character vector |
| collapse | logical; if TRUE, all extracted numbers per element of x are concatenated into a single string. Otherwise, a list of character vectors is returned. |
| output | character; controls the type of the returned values. "character" (default) returns strings; "numeric" coerces extracted values to numeric. |
| dec | character; decimal separator used in the input. Defaults to getOption("OutDec"). |

### Details

The function detects numeric values including optional signs, decimal parts, and scientific notation (e.g., `"1.23e-4"`).

Whitespace between sign and number (e.g., `"- 2.5"`) is tolerated and removed before returning results.

If `output = "numeric"` and the decimal separator `dec` differs from the current system setting (`getOption("OutDec")`), values are converted accordingly before coercion.

### Value

If `collapse = FALSE`, a list of character or numeric vectors containing the extracted values for each element of `x`. If `collapse = TRUE`, a character or numeric vector with concatenated values per element.

### See Also

`as.numeric()`, `stringi::stri_extract_all_regex()`

string-overview for an overview of all string utilities in pharos.

### Examples

```
x <- c("value = 12.5", "x = -3.2e2 and 7", "no numbers here")

# extract as character
strVal(x)

# extract as numeric
strVal(x, output = "numeric")

# concatenate values
strVal(x, collapse = TRUE)

# different decimal separator
strVal("value = 3,14", dec = ",", output = "numeric")
```

---

## `style` *Format Styles*

---

### Description

Interface for format templates, defined as a list consisting of any accepted format features in `fm()`. This enables to define templates globally and easily change or modify them later.

## Usage

```
styles()

style(
  x,
  digits = NULL,
  leadDigits = NULL,
  sci = NULL,
  bigMark = NULL,
  decMark = NULL,
  naForm = NULL,
  zeroForm = NULL,
  fmt = NULL,
  pThreshold = NULL,
  width = NULL,
  align = NULL,
  lang = NULL,
  label = NULL,
  ...
)

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

## Arguments

| | |
|---|---|
| x | an object of class Style or the name of a style, defined either in the global environment or in the options. The format arguments supplied along with it overwrite its settings or are added to them. If no style with the given name is found, a warning is issued and an empty style is returned. |
| digits | integer, the desired (fixed) number of digits after the decimal point. You will always get this number of digits, even if the last digit is 0. In scientific notation it is the number of decimals of the mantissa. Negative numbers of digits round to a power of ten (digits=-2 would round to the nearest hundred) for standard numeric formats; engineering formats require nonnegative values |
| leadDigits | number of leading zeros. leadDigits=3 would make sure that at least 3 digits on the left side will be printed, say 3.4 will be printed as 003.4. Setting leadDigits to 0 will yield results like .452 for 0.452. The default NULL will leave the numbers as they are (meaning at least one 0 digit). |
| sci | numeric scalar giving the absolute power-of-ten threshold for scientific notation. It applies to the absolute value of x and is used symmetrically: for sci = 8, a value is displayed scientifically if $\|x\| \geq 10^8$ or $0 < \|x\| < 10^{-8}$. The default is based on getOption("scipen"); an option value of zero is replaced by 7 |
| bigMark | character; if not empty used as mark between every 3 digits before the decimal point. If NULL, the option bigMark is used and, if that is not set, "" (none). |
| decMark | character specifying the decimal mark. If NULL, the current OutDec option is used |

| | |
|---|---|
| naForm | character, string specifying how NAs should be specially formatted. If set to NULL (default) no special action will be taken. |
| zeroForm | character, string specifying how zeros should be specially formatted. Useful for pretty printing 'sparse' objects. If set to NULL (default) no special action will be taken. |
| fmt | a format code or date-time template, or a formatting function. See the Details of fm(). |
| pThreshold | positive numeric threshold below which p-values are shown as "< threshold" |
| width | nonnegative integer giving the minimum display width |
| align | the character on whose position the strings will be aligned. Left alignment can be requested by setting align = "\\l", right alignment by "\\r" and center alignment by "\\c". Mind the backslashes, as if they are omitted, strings would be aligned to the **character** l, r or c respectively. The default is NULL which would just leave the strings as they are.<br>This argument is send directly to the function strAlign() as argument sep. |
| lang | optional value setting the language for the months and daynames. Can be either "local" for current locale or "en" for english. If left to NULL, the option DescToolsX.lang is used, falling back to "en" |
| label | a description for the style |
| ... | for style(), further components of the style: only arguments of fm() and the metadata name are accepted, anything else is an error. Not used by the print method. |

## Details

style() takes any of the arguments of fm() and combines them to an object of class "Style", which then can be handed over to fm() as argument fmt.
Following will define a new format template named "num.sty". Passed to fm() this will result in a number displayed with 2 fixed digits and a comma as big mark:

```
num.sty <- style(digits=2, bigMark=",")
fm(12222.89345, fmt=num.sty) = 12,222.89
```

This is the same result as if the arguments would have been supplied directly, but helps to avoid boilerplate code:
fm(12222.89345, digits=2, bigMark=",").

To derive a style from an existing one we can provide style() with the style, or with its name, and overwrite, resp. add new format options. style("num.sty", digits=1, sci=10) will use the current version of the numeric format and change the digits to 1 and the threshold to switch to scientific presentation to numbers >= 1e10 and < 1e-10. The stored style itself is not changed, so the result has to be assigned.

styles() returns all found style definitions in the global environment or in the options.

The styles can be stored as options for convenience. To store a new format we use the default options() approach: options(num.sty = style(digits=1, bigMark=" ")) Defined styles in the options can be passed on to fm() simply by their name. If a style of the same name exists in the global environment, that one is used.

Four default styles are registered as options when **pharos** is loaded, unless options of these names already exist: `"abs.sty"` for counts, `"num.sty"` for numeric values, `"per.sty"` for percentages and `"pval.sty"` for p-values. They are used by several plot functions of this package (e.g. `plotHeatmap()`) and by the report functions in **DescToolsX** (e.g. `DescToolsX::tOne()`). Redefining them with `options()` changes these formats globally.

## Value

`style()` returns an object of class `Style`
`styles()` returns a list of styles

## See Also

theme

Other format: `convUnit()`, `fm()`, `fmCI()`, `print.Unit()`, `unit()`

## Examples

```
# use style() to get and define new formats stored as option
num.sty <- style(digits=2, bigMark=" ")
abs.sty <- style(digits=0, bigMark=" ")
dat.sty <- style(fmt="MM, dd yyyy")

num.sty                              # displays the details of the style
# editing styles
style("abs.sty")                     # looks for format "abs.sty"
# style("nexist")                      # return for nonexisting style
style("abs.sty", bigMark="")       # get Style("abs") and overwrite bigMark
style("abs.sty", naForm="-")       # get Style("abs") and add user defined naForm

styles()                             # all defined formats
styles()[c("num.sty", "abs.sty")]   # numeric and integer styles

# define totally new format and store as option
options(nob.sty=style(digits=5, naForm="nodat"))

# using styles
fm(314.1563, fmt=abs.sty)
fm(314.1563, fmt=num.sty)

fm(Sys.Date(), fmt=dat.sty)

# remove the option again
options(nob.sty=NULL)
```

`textLegend` *Direct Labels in the Right Margin*

## Description

Draw a legend in the right margin consisting of short line segments and text labels, placed next to the (typically last) values of the plotted series. This labels lines directly instead of using a boxed `legend()`, as commonly preferred for time series and profile plots.

## Usage

```
textLegend(
  y,
  labels = names(y),
  line = c(1, 1),
  width = 1,
  col = par("fg"),
  lty = par("lty"),
  lwd = par("lwd"),
  cex = par("cex"),
  main = NULL,
  minDist = NULL
)
```

## Arguments

| | |
|---|---|
| `y` | numeric vector with the vertical anchor positions in user coordinates, typically the last observed value of each series, in series (column) order. Sorting and collision handling are done internally. |
| `labels` | character vector of labels, parallel to `y`. Defaults to `names(y)`, or the series index if `y` is unnamed. |
| `line` | numeric vector of length 1 or 2 (recycled), in margin lines of side 4. The first element is the offset of the segments from the plot region, the second the gap between segments and labels. |
| `width` | length of the line segments in margin lines. Set to `NA` to suppress the segments; the labels are then placed directly at `line[1]`. |
| `col`, `lty`, `lwd` | color, line type and width of the segments, parallel to `y` (in series order, recycled). Ignored if `width` is `NA`. |
| `cex` | character expansion for the labels. Also enters the default vertical spacing, so larger text automatically gets wider spacing. |
| `main` | optional title for the legend, drawn at the top of the plot region. Default is `NULL` (none). |
| `minDist` | minimal vertical distance between labels in user coordinates, passed to `spreadOut()`. Default is `1.2 * strheight("M") * cex`. |

## Details

The function owns the legend geometry: the anchor positions `y` are sorted internally so the legend follows the vertical order of the lines, the graphical parameters are recycled accordingly, and vertical label collisions are resolved via `spreadOut()`. Callers therefore simply pass the values in series order, parallel to `col`, `lty` and `lwd`.

Drawing uses `mtext()` and `segments()` in the device margin; `xpd` is set to `TRUE` and restored on exit. Horizontal positions are given in margin lines of side 4 and converted to user coordinates via `lineToUser()`.

Make sure the right margin is wide enough for segments and labels, e.g. via the `mar` argument of the calling plot function.

## Value

The (sorted and spread) vertical label positions, invisibly. Useful for adding further annotation next to the labels.

## See Also

`plotLines()`, `mtext()`, `segments()`, `legend()`

Other graphics.annotation: `barText()`, `boxedText()`, `colLegend()`, `errBars()`, `stamp()`, `titleRect()`

## Examples

```
m <- EuStockMarkets[seq(1, 1860, 10), ]
op <- par(mar = c(5, 4, 4, 8))
matplot(m, type = "l", lty = 1, lwd = 2, col = 1:4, las = 1,
        main = "EU Stock Markets")
textLegend(y = m[nrow(m), ], labels = colnames(m),
           col = 1:4, lwd = 2, main = "Index")
par(op)
```

---

`theme` *pharos's Graphics and Formatting Theme*

---

## Description

`getTheme()`, `setTheme()`, and `resetTheme()` are the user-facing entry points to pharos's theme system: a single, named list of graphical and formatting defaults, consulted by essentially every plotting function in the package (and by `fm()` for numeric/percentage/p-value formatting) whenever the corresponding function argument is left at its default value.

## Usage

```
getTheme()

setTheme(theme)

resetTheme()
```

## Arguments

| | |
|---|---|
| theme | either a named list of theme components to merge into the active theme (only the given top-level elements are replaced; e.g. `setTheme(list(palette = "Set2"))` changes only the palette, leaving `grid`, `box`, `twin`, etc. untouched), or a single string naming a preset registered in the (currently empty) preset registry. |

## Value

`getTheme()` and `resetTheme()` return the (new) active theme, a named list; `setTheme()` returns the new active theme as well, invisibly.

## What the theme is for

Most graphical parameters in pharos's plotting functions (`col`, `grid`, `box`, `pch`, ...) default to a sentinel value, `.useTheme`, rather than to a hardcoded color or number. At call time, that sentinel is resolved against `getTheme()` - so changing the active theme changes the look of *every* plot produced afterwards that didn't explicitly override that argument, without touching a single plotting function's code.

This gives three independent ways to control a plot's appearance, in order of precedence (highest first):

1. **Explicit argument**, e.g. `plotXY(x, y, col = "red")` - always wins, for that one call only.
2. **Function-specific default** - some functions deliberately hardcode a value that differs from the generic theme baseline instead of using `.useTheme` (e.g. `plotBar()`'s `box` defaults to `FALSE` rather than the theme's box setting, and `plotDot()`'s grid line color/style stays its own orange/grey dashed look regardless of the active theme). This is a conscious per-function design choice, not a bug - see the individual function's documentation for which arguments opt out of theme-following this way.
3. **Active theme** (`getTheme()`) - the package-wide fallback used whenever neither of the above applies.

## Structure of the theme

The theme is a named list with the following top-level components. Nested components (`grid`, `box`, `points`, `bar`, `sty`) are themselves named lists; `setTheme()` replaces them wholesale (it does not merge one level deeper), so to change a single nested value, supply the complete sub-list, e.g. `setTheme(list(grid = list(col = "red", lwd = 1, lty = "dotted")))`.

`par` `col.axis="grey40", las=1, cex=1.1`. Global `par()` pass applied by every `.applyParFromDots()` call (axis label color, axis label orientation, global text scaling).

`grid col="grey80", lwd=1, lty="dotted"`, plus `group.*` variants. Background grid lines, via `.drawGrid()`. The `group.*` entries style a subordinate/secondary grid (e.g. group separators) where a function draws one.

`box col="grey50", lwd=1, lty="solid"`. The frame drawn around a plot region, via `.drawBox()`.

`points pch=21, col="grey50", bg=addOpacity("grey"), cex=1.1`. Default point styling for scatterplot-like functions (e.g. `plotXY()`, `plotDot()`).

`twin pal("helsana")[c(6, 1)]`. A fixed pair of colors for contexts that inherently need exactly two contrasting colors (e.g. a fit line and a smoother in `plotXY()`, the two poles of a diverging color ramp in `plotCor()`, a single accent color via `twin[1]` in `lines.loess()`/`plotQQ()`'s confidence band). Never used as a substitute for `palette` when more than two colors are needed.

`palette "helsana"`. Name of the qualitative (categorical) palette used whenever more than two unordered colors are needed (e.g. `plotMosaic()`'s fill colors), resolved via `pal()`. Deliberately not used for sequential or diverging numeric scales – see the next item.

**(none – by design)** Sequential/diverging numeric color scales (e.g. `plotDens2D()`'s density heatmap, `plotHeatmap()`'s cell shading) are deliberately *not* theme-driven; they use a hardcoded, purpose-built palette via `pal()` instead (e.g. `pal("red-black")`, `pal("Blues")`). Neither `palette` (categorical) nor `twin` (a fixed pair) is the right semantic fit for an ordered, continuous scale – see `pal()` for the registry of named continuous palettes.

`bar col="grey80", border=NA`. Default bar fill/border in `plotBar()`.

`autoTitle FALSE`. With `TRUE`, plots called with `main = NULL` (the default) get a title derived from the input, e.g. `"len ~ supp"` for `plotBox(len ~ supp, ToothGrowth)`; with `FALSE` they get none. An explicit `main` always wins. Handy for interactive exploration or teaching: `setTheme(list(autoTitle = TRUE))`.

`sty abs="abs.sty", perc="per.sty", num="num.sty", pval="pval.sty"`. Names of `fm()` format styles (see `styles()`) used for absolute counts, percentages, plain numbers, and p-values respectively.

`stamp expression(...)` – unevaluated. The corner stamp text drawn by every `stamp()`/`.withGraphicsState()` call. Stored as an unevaluated `expression()` and `eval()`'d at draw time (not at theme-load or theme-set time), so it always reflects the current user and date rather than freezing whatever they were when the theme was defined or last changed. Default: `"<username> / <YYYY-MM-DD>"`.

### The `.useTheme` sentinel

Internally, a function argument that should follow the active theme (rather than some fixed value) defaults to a dedicated sentinel object, `.useTheme`, not to `TRUE`, `FALSE`, `NA`, or `NULL`. Those four are frequently legitimate, explicit values in their own right (e.g. `grid = NULL` commonly means "suppress the grid" for a given function) - using any of them to also mean "no value was given, defer to the theme" would make that case ambiguous. A dedicated sentinel avoids the ambiguity entirely and keeps the resolution logic a single, explicit equality check (`identical(x, .useTheme)`) rather than an implicit, error-prone guess based on data type.

Two small internal helpers resolve it:

- `.useThemeValue(value, ...)` - for a simple value taken from a nested theme key, e.g. `col <- .useThemeValue(col, "points", "col")`.

- `.resolveToggle(spec, themeValue)` - for an on/off-style argument (such as `grid`/`box`) that may also be a `TRUE`/`FALSE`/`NA`/`NULL`/list spec in its own right, used internally by `.drawGrid()`/`.drawBox()`.

Some theme values require more than a simple key lookup to resolve (e.g. building a color ramp from `twin`, or constructing a list of point parameters from `points`); those are resolved with a small inline `identical(x, .useTheme)` check directly in the consuming function rather than forcing them through one of the two generic helpers above. See e.g. `plotCor()`'s `col` argument or `plotDot()`'s `pch` argument for worked examples.

## How plotting functions consume the theme

`.applyParFromDots()` Applies `getTheme()$par` as the lowest-precedence `par()` pass, before the function's own defaults and the user's `...`.

`.drawGrid(grid, defaults = list())` / `.drawBox(box, defaults = list())` Generic dispatchers wrapping `graphics::grid()`/`graphics::box()`. Resolve the `.useTheme` sentinel, merge the theme's style values with any function-specific `defaults` (e.g. `plotBar()` suppressing the axis-parallel grid direction via `nx`/`ny`), and dispatch via `bedrock::callIf()`. **Not** used by every function that draws a grid or frame: a few (`plotCor()`, `plotHeatmap()`, `plotDot()`) have grid/box geometry tied to exact data coordinates (e.g. half-integer cell boundaries) that doesn't match `graphics::grid()`'s axis-tick-based geometry: these resolve theme values directly via `getTheme()$grid`/`getTheme()$box` but draw with their own clipped `rect()`/`abline()` calls instead.

`.withGraphicsState(expr, stamp = .useTheme, ...)` Wraps a plotting function's body, restores `par()` afterwards, and - after `expr` has run successfully - calls `stamp()` with the (possibly theme-resolved) corner stamp text/arguments.

## Presets

`setTheme()` also accepts a single preset name (a string) instead of a list, looked up in an internal preset registry (`.themePresets`). No presets are currently registered; the registry exists so named, complete theme variants (e.g. a monochrome or high-contrast theme) can be added later without changing the `setTheme()` interface.

## See Also

`pal()` for the color palette registry, `fm()` for the formatting styles referenced by `sty`, `stamp()` for the corner stamp mechanism.

## Examples

```
# inspect the active theme
getTheme()

# change only the qualitative palette for the rest of the session
setTheme(list(palette = "Set2"))

# change the accent color pair used for e.g. lm/loess fit lines
setTheme(list(twin = c("firebrick", "navy")))

# turn grid lines off package-wide (any .useTheme-driven grid argument
```

```
# everywhere now resolves to "off", unless a function overrides it)
setTheme(list(grid = FALSE))

# back to package defaults
resetTheme()
```

---

`titleRect` *Plot Boxed Annotation*

---

## Description

The function can be used to add a title to a plot surrounded by a rectangular box. This is useful for plotting several plots in narrow distances.

## Usage

```
titleRect(
  label,
  bg = "grey",
  border = 1,
  col = "black",
  xjust = 0.5,
  line = 2,
  ...
)
```

## Arguments

| | |
|---|---|
| `label` | the main title |
| `bg` | the background color of the box. |
| `border` | the border color of the box |
| `col` | the font color of the title |
| `xjust` | the x-justification of the text. This can be `c(0, 0.5, 1)` for left, middle- and right alignement. |
| `line` | on which MARgin line, starting at 0 counting outwards |
| `...` | the dots are passed to the `text()` function, which can be used to change font and similar arguments. |

## Value

nothing is returned

## See Also

`title()`

Other graphics.annotation: `barText()`, `boxedText()`, `colLegend()`, `errBars()`, `stamp()`, `textLegend()`

## Examples

```
plot(pressure)
titleRect("pressure")
```

---

`toHtmlTable` *Render a matrix as an HTML table*

---

## Description

Converts a matrix (or vector) to a `<table>` HTML fragment, with optional row/column headers, caption, per-column alignment and widths. The result has class `c("html", "character")` (see `asHtml()`) and prints as a formatted text table via `preview.html()`.

## Usage

```
toHtmlTable(
  m,
  sepCol = FALSE,
  caption = "",
  bodyAlign = "center",
  vAlign = "top",
  width = NULL,
  cellPadding = 3,
  border = 0,
  tableWidth = NA,
  captionAlign = "center",
  frame = TRUE,
  rowNames = TRUE,
  colNames = TRUE
)
```

## Arguments

| | |
|---|---|
| `m` | a matrix or vector |
| `sepCol` | logical; if `TRUE`, insert a narrow empty separator column between each pair of columns |
| `caption` | table caption text |
| `bodyAlign` | horizontal alignment of body cells (`"left"`, `"center"`, `"right"`), recycled to the number of columns |
| `vAlign` | vertical alignment of body cells (HTML `valign` attribute: `"top"`, `"middle"`, `"bottom"`), recycled to the number of columns |
| `width` | column width(s) (HTML `width` attribute), recycled to the number of columns including an optional rowname column; use `NA` for columns without an explicit width |

| | |
|---|---|
| cellPadding | HTML cellpadding attribute |
| border | HTML border attribute |
| tableWidth | overall table width (HTML width attribute on <table>), or NA for none |
| captionAlign | horizontal alignment of the header row cells |
| frame | logical; if TRUE, draw outer frame and group rules (frame="hsides" rules="groups") |
| rowNames | logical; render the row names as a leading header column. Ignored when m has none |
| colNames | logical; render the column names as a header row. Ignored when m has none |

## Value

an object of class c("html", "character")

## See Also

bedrock::appendEnum

Other html: asFileLink(), asHtml(), asImg(), embedFile(), escapeHtml(), htmlNotation, htmlSubscript, preview(), preview.html()

## Examples

```
m <- matrix(c(12.3, 4.56, 7.8, 91.2), nrow = 2,
            dimnames = list(c("a", "b"), c("mean", "sd")))

# the HTML fragment ...
(h <- toHtmlTable(m, caption = "Summary"))

# ... and its rendering as text
preview(h)

# formatted cells, right aligned, without the row names
preview(toHtmlTable(fm(m, digits = 1), rowNames = FALSE, bodyAlign = "right"))
```

---

# transformXY *Apply Geometric Transformations to Coordinates*

---

## Description

Applies scaling, rotation, and translation to a set of 2D coordinates. The transformations are applied in the following order:

1. Scaling
2. Rotation (see rotate())
3. Translation

## Usage

```
transformXY(
  x,
  y = NULL,
  translate = c(0, 0),
  scale = c(1, 1),
  theta = 0,
  asp = 1
)
```

## Arguments

| | |
|---|---|
| x | numeric vector of x coordinates, or an object coercible by xy.coords(). |
| y | numeric vector of y coordinates. Ignored if x already contains both coordinates. |
| translate | numeric vector of length 1 or 2 specifying translation in x and y direction. Recycled if necessary. Default is c(0, 0). |
| scale | numeric vector of length 1 or 2 specifying scaling factors for x and y. Recycled if necessary. Default is c(1, 1). |
| theta | rotation angle in radians. Default is 0. |
| asp | aspect ratio adjustment passed to rotate(). Default is 1. |

## Details

This function is a convenience wrapper combining basic affine transformations. Internally, it uses rotate() for rotation.

## Value

A list with components x and y, as returned by xy.coords().

## See Also

Other geometry.transformation: rotate()

## Examples

```
x <- c(0, 1, 1, 0)
y <- c(0, 0, 1, 1)

# scale, rotate and translate a square
transformXY(x, y,
            scale = 2,
            theta = pi / 4,
            translate = c(1, 1))

# matrix input
m <- cbind(x, y)
transformXY(m, scale = 0.5, theta = pi/6)
```

unit *Get or Set Unit Attribute*

## Description

Retrieves or assigns a unit attribute to an R object.

## Usage

```
unit(x)

unit(x) <- value
```

## Arguments

| | |
|---|---|
| x | an R object. |
| value | A single character string specifying the unit, or NULL to remove the unit attribute. |

## Details

The unit is stored as a simple character string in the "unit" attribute. This provides a lightweight mechanism for attaching unit metadata to objects, without enforcing any automatic conversion or validation.

The setter does not validate whether the unit is physically meaningful. Validation and conversion should be handled externally (e.g., via a unit conversion engine such as ConvUnit7).

Assigning NULL removes the unit attribute.

## Value

- unit(x) returns the unit as a character string or NULL.
- unit(x) <- value returns x with updated unit attribute.

## See Also

Other format: convUnit(), fm(), fmCI(), print.Unit(), style()

## Examples

```
x <- 10
unit(x) <- "m"
unit(x)

y <- 5
unit(y) <- "kg"

# remove unit
```

```
unit(y) <- NULL
```

# Index


∗ **agreement**
plot.BlandAltman, 78
plotCor, 111
∗ **annotation**
abcCoords, 5
axisBreak, 15
barText, 20
boxedText, 24
colLegend, 30
errBars, 44
lines.lm, 68
lines.loess, 70
polarGrid, 175
splineCI, 190
stamp, 193
textLegend, 219
titleRect, 224
∗ **attribute**
print.Unit, 181
unit, 228
∗ **bar-chart**
plotBar, 97
∗ **base-graphics**
plotBubble, 105
plotDens, 115
plotDens2D, 117
plotRidge, 156
∗ **binning**
findCol, 51
∗ **bivariate**
plotAssoc, 90
plotBag, 93
plotBubble, 105
plotCor, 111
plotDens2D, 117
plotHeatmap, 136
plotHexbin, 138
plotMosaic, 146
plotPolar, 148
plotTernary, 162
plotWeb, 170
plotXY, 172
∗ **boxplot**
plotBox, 102
plotDensBox, 119
∗ **classification**
contrastCol, 37
∗ **color-conversion**
cmykToCmy, 28
cmykToRgb, 28
cmyToCmyk, 29
colToHex, 34
colToHsv, 34
colToRgb, 36
grayScale, 61
hexToCol, 63
hexToRgb, 64
longToRgb, 73
rgbToCmy, 183
rgbToCol, 183
rgbToHex, 184
rgbToLong, 185
∗ **color.lookup**
contrastCol, 37
findCol, 51
∗ **color.manipulation**
addOpacity, 7
colToOpaque, 35
darken, 41
fade, 50
lighten, 67
mixCol, 75
∗ **color.palettes**
hcol, 62
pal, 76
palNames, 77
∗ **color**
addOpacity, 7

cmykToCmy, 28
cmykToRgb, 28
cmyToCmyk, 29
colToHex, 34
colToHsv, 34
colToOpaque, 35
colToRgb, 36
contrastCol, 37
darken, 41
fade, 50
findCol, 51
grayScale, 61
hcol, 62
hexToCol, 63
hexToRgb, 64
lighten, 67
longToRgb, 73
mixCol, 75
pal, 76
palNames, 77
rgbToCmy, 183
rgbToCol, 183
rgbToHex, 184
rgbToLong, 185
setBackCol, 188

∗ **composition**
plotSankey, 159

∗ **confidence-interval**
as.CI, 9
errBars, 44
fmCI, 59
plotDot, 122
plotPropCI, 152

∗ **correlation**
plotCor, 111

∗ **datasets**
Prefix, 178

∗ **date-time**
axTicksPosixct, 18

∗ **density-estimation**
plotDens2D, 117

∗ **density**
plotDens, 115
plotDensBox, 119
plotRidge, 156
plotViolin, 166

∗ **descriptive-statistics**
plotDens2D, 117

∗ **distribution-summary**
plot.Desc.qn, 80
plotBox, 102
plotDens, 115
plotECDF, 125
plotFun, 134
plotProbDist, 150
plotQQ, 154
plotRidge, 156
plotViolin, 166

∗ **dotchart**
plotDot, 122

∗ **facet**
plotFacet, 128

∗ **flow-diagram**
plotCirc, 109
plotSankey, 159

∗ **formatting**
asHtml, 13
asImg, 14
fm, 53
fmCI, 59
htmlNotation, 64
preview, 179
preview.html, 180
print.Unit, 181
strAbbr, 194
strAlign, 195
strCap, 196
strPad, 207
strTrim, 212
strTrunc, 213
style, 215
unit, 228

∗ **format**
convUnit, 38
fm, 53
fmCI, 59
print.Unit, 181
style, 215
unit, 228

∗ **frequency-table**
ftable.list, 60
plot.Desc.table, 82
plotAssoc, 90
plotCatDist, 107
plotCirc, 109
plotFreqDist, 131

plotMosaic, 146
plotTreemap, 164
∗ **geometry.conversion**
coordinate-conversions, 40
degree-radians-conversion, 42
∗ **geometry.structures**
arc, 8
band, 19
bezier, 23
circle, 27
ellipse, 43
polygonX, 177
regPolygon, 182
ring, 186
∗ **geometry.transformation**
rotate, 187
transformXY, 226
∗ **geometry**
abcCoords, 5
arc, 8
band, 19
bezier, 23
canvas, 26
circle, 27
coordinate-conversions, 40
degree-radians-conversion, 42
ellipse, 43
isValidPlotRegion, 66
lineToUser, 72
mar, 74
polarGrid, 175
polygonX, 177
regPolygon, 182
ring, 186
rotate, 187
setBackCol, 188
shade, 189
transformXY, 226
∗ **graphics-export**
exportPlot, 48
∗ **graphics.annotation**
barText, 20
boxedText, 24
colLegend, 30
errBars, 44
stamp, 193
textLegend, 219
titleRect, 224
∗ **graphics.framework**
exportPlot, 48
∗ **graphics.layout**
abcCoords, 5
axisBreak, 15
axTicksPosixct, 18
isValidPlotRegion, 66
lineToUser, 72
mar, 74
plotFacet, 128
spreadOut, 192
∗ **graphics.setup**
canvas, 26
polarGrid, 175
setBackCol, 188
∗ **graphics.trendlines**
lines.lm, 68
lines.loess, 70
splineCI, 190
∗ **graphics**
plot.Lc, 85
plotDens2D, 117
∗ **html**
asFileLink, 12
asHtml, 13
asImg, 14
embedFile, 44
escapeHtml, 47
htmlNotation, 64
htmlSubscript, 65
preview, 179
preview.html, 180
toHtmlTable, 225
∗ **inequality**
plot.Lc, 85
∗ **kernel-methods**
plotDens2D, 117
∗ **label**
abcCoords, 5
axisBreak, 15
axTicksPosixct, 18
barText, 20
boxedText, 24
colLegend, 30
spreadOut, 192
stamp, 193
textLegend, 219
titleRect, 224

∗ **line-chart**
plotArea, 88
plotLift, 140
plotLines, 143
∗ **method-comparison**
plot.BlandAltman, 78
∗ **missing-value**
plotMiss, 145
∗ **model-evaluation**
plotLift, 140
∗ **normality-test**
plotQQ, 154
∗ **number-formatting**
fm, 53
strVal, 214
style, 215
∗ **numerical-methods**
arc, 8
band, 19
bezier, 23
canvas, 26
circle, 27
coordinate-conversions, 40
degree-radians-conversion, 42
ellipse, 43
lineToUser, 72
mar, 74
polygonX, 177
regPolygon, 182
ring, 186
rotate, 187
spreadOut, 192
strDist, 199
transformXY, 226
∗ **palette**
hcol, 62
pal, 76
palNames, 77
∗ **panel**
plotFacet, 128
∗ **pattern-matching**
strExtract, 201
strExtractBetween, 202
strPos, 208
strSplit, 211
strVal, 214
∗ **phonetic-encoding**
strSpell, 210
∗ **plot.bivariate**
plotAssoc, 90
plotBag, 93
plotBubble, 105
plotCor, 111
plotDens2D, 117
plotHeatmap, 136
plotHexbin, 138
plotMosaic, 146
plotXY, 172
∗ **plot.distribution**
plotFun, 134
plotProbDist, 150
shade, 189
∗ **plot.s3**
plot.BlandAltman, 78
plot.Desc.qn, 80
plot.Desc.table, 82
plot.Lc, 85
∗ **plot.special**
plotBinaryTree, 100
plotCirc, 109
plotLift, 140
plotMiss, 145
plotPolar, 148
plotPropCI, 152
plotSankey, 159
plotTernary, 162
plotTimeSeries, 163
plotTreemap, 164
plotWeb, 170
∗ **plot.univariate**
plotArea, 88
plotBar, 97
plotBox, 102
plotCatDist, 107
plotDens, 115
plotDensBox, 119
plotDot, 122
plotECDF, 125
plotFreqDist, 131
plotLines, 143
plotQQ, 154
plotRidge, 156
plotViolin, 166
∗ **plotting**
plotBubble, 105
plotDens, 115

plotRidge, 156
∗ **prediction**
plotLift, 140
∗ **proportion**
plotPropCI, 152
∗ **regression**
lines.lm, 68
lines.loess, 70
splineCI, 190
∗ **reproducibility**
exportPlot, 48
∗ **reshape**
strChop, 197
strLeftRight, 205
strSplit, 211
∗ **robust-statistics**
plotBag, 93
∗ **scatterplot**
plotBubble, 105
plotHexbin, 138
plotXY, 172
∗ **string-inspection**
strCountWords, 199
strDist, 199
strIsNumeric, 204
strLen, 206
strPos, 208
∗ **string-manipulation**
lineSep, 71
strAbbr, 194
strAlign, 195
strCap, 196
strChop, 197
strExtract, 201
strExtractBetween, 202
strLeftRight, 205
strPad, 207
strRev, 209
strSpell, 210
strSplit, 211
strTrim, 212
strTrunc, 213
strVal, 214
∗ **summary**
strCountWords, 199
strLen, 206
∗ **symbolic-units**
convUnit, 38
∗ **tables**
ftable.list, 60
∗ **theme**
theme, 220
∗ **time-series**
plotLines, 143
plotTimeSeries, 163
∗ **transformation**
addOpacity, 7
colToOpaque, 35
darken, 41
fade, 50
lighten, 67
mixCol, 75
∗ **tree**
plotBinaryTree, 100
∗ **type-test**
strIsNumeric, 204
∗ **unit-conversion**
convUnit, 38
%_% *(*htmlSubscript*)*, 65

abcCoords, 5
abcCoords(), *16*, *19*, *66*, *72*, *74*, *130*, *193*
abline(), *69*, *129*
addOpacity, 7
addOpacity(), *36*, *41*, *51*, *67*, *75*
arc, 8
arc(), *20*, *23*, *27*, *43*, *178*, *182*, *186, 187*
arrows(), *45, 46*
as.CI, 9
as.CI(), *123, 124*
as.numeric(), *205*, *215*
asFileLink, 12
asFileLink(), *13, 14*, *44*, *47*, *65*, *179, 180*, *226*
asHtml, 13
asHtml(), *12*, *14*, *44*, *47*, *65*, *179, 180*, *225*, *226*
asImg, 14
asImg(), *12, 13*, *44*, *47*, *65*, *179, 180*, *226*
axis, *15*
axisBreak, 15
axisBreak(), *6*, *19*, *66*, *72*, *74*, *130*, *193*
axisFmt, 16
axTicksDate *(*axTicksPosixct*)*, 18
axTicksPosixct, 18
axTicksPosixct(), *6*, *16*, *66*, *72*, *74*, *130*, *193*

band, 19
band(), *9*, *23*, *27*, *43*, *178*, *182*, *186*
barplot(), *21*
barText, 20, *99*
barText(), *26*, *31*, *46*, *98*, *99*, *194*, *220*, *224*
base::attr, *181*
base::format, *58*
base::formatC, *58*
base::match.arg(), *76*
base::nchar(), *206*
base::prettyNum, *58*
base::sprintf, *58*
base::strptime, *18*
base::strsplit(), *211*
base::substr(), *204*
base::Sys.setlocale, *58*
base::trimws(), *212*
bedrock::appendEnum, *226*
bedrock::callIf, *129*, *130*
bedrock::callIf(), *104*, *121*, *133*, *142*, *174*, *223*
bedrock::countCompCases(), *146*
bedrock::label, *181*
bedrock::resolveFormula(), *104*, *116*, *117*, *121*, *127*, *158*, *168*
bezier, 23
bezier(), *9*, *20*, *27*, *43*, *178*, *182*, *186*
binaryTree *(*plotBinaryTree*)*, 100
box(), *151*
boxedText, 24
boxedText(), *21*, *22*, *31*, *46*, *101*, *151*, *194*, *220*, *224*
boxplot(), *104*, *116*, *121*, *127*, *133*, *158*, *168*, *169*

call(), *189*
canvas, 26
canvas(), *101*, *176*, *188*
cartToPol *(*coordinate-conversions*)*, 40
cartToSph *(*coordinate-conversions*)*, 40
circle, 27
circle(), *9*, *20*, *23*, *43*, *177*, *178*, *182*, *186*
cmykToCmy, 28
cmykToCmy(), *33*
cmykToRgb, 28
cmykToRgb(), *33*
cmyToCmyk, 29
cmyToCmyk(), *33*
colLegend, 30
colLegend(), *22*, *26*, *46*, *194*, *220*, *224*
color-conversion-overview, *28*, *29*, 32, *34–36*, *62–64*, *73*, *183–185*
colToHex, 34
colToHex(), *32*, *33*
colToHsv, 34
colToHsv(), *32*
colToOpaque, 35
colToOpaque(), *7*, *33*, *41*, *51*, *67*, *75*
colToRgb, 36
colToRgb(), *32*, *33*, *184*, *185*
constants *(*Prefix*)*, 178
contrastCol, 37
contrastCol(), *52*
convUnit, 38, *178*
convUnit(), *58*, *59*, *181*, *218*, *228*
coordinate-conversions, 40
curve(), *150*, *151*, *189*

darken, 41
darken(), *7*, *36*, *51*, *67*, *75*
degree-radians-conversion, 42
degToRad *(*degree-radians-conversion*)*, 42
degToRad(), *40*
density(), *133*, *169*
dev.cur(), *66*
dotchart(), *45*

ecdf(), *127*
ellipse, 43
ellipse(), *9*, *20*, *23*, *27*, *177*, *178*, *182*, *186*, *187*
embedFile, 44
embedFile(), *12–14*, *47*, *65*, *179*, *180*, *226*
errBars, 44
errBars(), *22*, *26*, *31*, *194*, *220*, *224*
escapeHtml, 47
escapeHtml(), *12–14*, *44*, *65*, *179*, *180*, *226*
exportPlot, 48
expression(), *189*

fade, 50
fade(), *7*, *36*, *41*, *62*, *67*, *75*
findCol, 51
findCol(), *38*
findInterval(), *52*
fm, *18*, 53, *178*
fm(), *16*, *17*, *39*, *59*, *160*, *181*, *217*, *218*, *220*, *222*, *223*, *228*

fmCI, 59
fmCI(), *10*, *39*, *58*, *181*, *218*, *228*
ftable(), *60*
ftable.list, 60

getTheme *(*theme*)*, 220
getTheme(), *148*
graphics::abline(), *86*, *113*, *141*
graphics::axis, *18*
graphics::axis(), *16*, *17*, *112*, *144*
graphics::axis.POSIXct(), *19*
graphics::axTicks, *18*
graphics::axTicks(), *17–19*, *129*
graphics::barplot, *99*
graphics::barplot(), *97–99*, *153*
graphics::box(), *81*, *84*, *86*, *118*, *127*, *137*, *153*, *155*, *173*
graphics::boxplot(), *103*, *104*, *121*, *132*
graphics::cdplot(), *81*, *82*, *117*
graphics::dotchart(), *124*
graphics::grid(), *86*, *89*, *98*, *103*, *113*, *118*, *121*, *127*, *135*, *139*, *141*, *144*, *153*, *155*, *173*
graphics::hist(), *132*
graphics::image(), *111–113*, *138*
graphics::layout, *130*
graphics::legend, *31*
graphics::legend(), *81*, *98*, *116*, *127*, *141*, *153*, *171*, *174*
graphics::lines, *23*
graphics::lines(), *87*, *141*, *173*
graphics::matplot(), *143*
graphics::mosaicplot, *92*
graphics::par(), *74*, *87*, *89*, *98*, *112*, *120*, *123*, *135*, *137*, *144*, *153*, *171*, *176*, *193*
graphics::plot(), *17*, *81*, *174*
graphics::points(), *87*
graphics::polygon, *20*
graphics::polygon(), *177*, *189*
graphics::polypath(), *178*
graphics::rect(), *91*
graphics::rug(), *132*
graphics::spineplot(), *81–84*, *148*
graphics::text, *18*
graphics::text(), *17*, *160*
grayScale, 61
grayScale(), *33*
grDevices::adjustcolor, *7*
grDevices::adjustcolor(), *76*, *77*
grDevices::col2rgb, *36*
grDevices::col2rgb(), *32*, *33*
grDevices::colorRampPalette(), *76*, *77*
grDevices::colors(), *183*
grDevices::convertColor(), *184*
grDevices::hsv(), *33*
grDevices::png(), *14*
grid(), *150*, *176*
grid::grid.draw(), *49*

hclust(), *146*
hcol, 62
hcol(), *77*
hexToCol, 63
hexToCol(), *32*
hexToRgb, 64
hexToRgb(), *32*
hist(), *133*
htmlBar *(*htmlNotation*)*, 64
htmlHat *(*htmlNotation*)*, 64
htmlNotation, *12–14*, *44*, *47*, 64, *65*, *179*, *180*, *226*
htmlSubscript, *12–14*, *44*, *47*, *65*, 65, *179*, *180*, *226*

isCI *(*as.CI*)*, 9
isCI(), *124*
isValidPlotRegion, 66
isValidPlotRegion(), *6*, *16*, *19*, *72*, *74*, *130*, *193*

layout(), *121*, *130*, *132*, *133*
legend(), *5*, *6*, *219*, *220*
lighten, 67
lighten(), *7*, *36*, *41*, *51*, *75*
lines(), *69*, *70*, *190*, *191*
lines.Lc *(*plot.Lc*)*, 85
lines.LcList *(*plot.Lc*)*, 85
lines.lm, 68
lines.lm(), *71*, *191*
lines.lmlog *(*lines.lm*)*, 68
lines.loess, 70, *156*
lines.loess(), *69*, *127*, *135*, *191*, *222*
lines.SplineX *(*splineCI*)*, 190
lines.splineX *(*splineCI*)*, 190
lineSep, 71
lineToUser, 72
lineToUser(), *6*, *16*, *19*, *66*, *74*, *130*, *193*

lm(), *69*
loess(), *70, 71*, *191*
longToRgb, 73
longToRgb(), *33*

mar, 74
mar(), *6*, *16*, *19*, *66*, *72*, *130*, *193*
MASS::fractions(), *57*
mixCol, 75
mixCol(), *7*, *36*, *41*, *51*, *67*
mtext(), *72*, *151*, *220*

na.omit(), *104*, *106*, *116*, *121*, *158*, *168*
nchar(), *206*

pal, 76
pal(), *63*, *77*, *222, 223*
palNames, 77
palNames(), *63*, *76, 77*
par(), *21*, *66*, *76*, *81*, *84*, *110*, *126*, *133*, *160*
plot(), *146*
plot.BlandAltman, 78
plot.BlandAltman(), *82*, *84*, *87*
plot.Desc.qn, 80
plot.Desc.qn(), *79*, *84*, *87*
plot.Desc.table, 82
plot.Desc.table(), *79*, *82*, *87*
plot.ecdf(), *127*
plot.formula(), *94*, *174*
plot.Lc, 85
plot.Lc(), *79*, *82*, *84*
plot.LcList *(*plot.Lc*)*, 85
plot.new(), *130*
plot.Palette *(*pal*)*, 76
plot.window(), *26*
plotArea, 88
plotArea(), *99*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *156*, *158*, *169*
plotAssoc, 90
plotAssoc(), *83, 84*, *96*, *107*, *113*, *119*, *138, 139*, *148*, *174*
plotBag, 93
plotBag(), *92*, *107*, *113*, *119*, *138, 139*, *148*, *174*
plotBar, 97
plotBar(), *89*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *148*, *156*, *158*, *169*
plotBinaryTree, 100
plotBinaryTree(), *111*, *142*, *146*, *149*, *153*, *161*, *163, 164*, *166*, *171*
plotBox, 102
plotBox(), *81, 82*, *89*, *99*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *156*, *158*, *169*
plotBubble, 105
plotBubble(), *92*, *96*, *113*, *119*, *138, 139*, *148*, *174*
plotCatDist, 107
plotCatDist(), *89*, *99*, *104*, *117*, *121*, *124*, *127*, *133*, *144*, *148*, *156*, *158*, *169*
plotCirc, 109
plotCirc(), *101*, *142*, *146*, *149*, *153*, *161*, *163, 164*, *166*, *171*
plotCor, 111
plotCor(), *92*, *96*, *107*, *119*, *138, 139*, *148*, *153*, *170, 171*, *174*
plotDens, 115
plotDens(), *81, 82*, *89*, *99*, *104*, *109*, *121*, *124*, *127*, *133*, *144*, *156*, *158*, *163*, *169*
plotDens2D, 117
plotDens2D(), *92*, *96*, *107*, *113*, *138, 139*, *148*, *153*, *174*
plotDensBox, 119
plotDensBox(), *89*, *99*, *104*, *109*, *117*, *124*, *127*, *133*, *144*, *156*, *158*, *169*
plotDot, 122
plotDot(), *10*, *89*, *99*, *104*, *109*, *117*, *121*, *127*, *133*, *144*, *156*, *158*, *169*
plotECDF, 125
plotECDF(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *132, 133*, *141, 142*, *144*, *156*, *158*, *169*
plotFacet, 128
plotFacet(), *6*, *16*, *19*, *66*, *72*, *74*, *193*
plotFreqDist, 131
plotFreqDist(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *127*, *144*, *156*, *158*, *169*
plotFun, 134
plotFun(), *151*, *189*
plotHeatmap, 136
plotHeatmap(), *83, 84*, *92*, *96*, *107*, *113*, *119*, *139*, *148*, *153*, *174*, *218*
plotHexbin, 138
plotHexbin(), *92*, *96*, *107*, *113*, *119*, *138*, *148*, *174*
plotLift, 140

plotLift(), *101*, *111*, *146*, *149*, *153*, *161*, *163*, *164*, *166*, *171*
plotLines, 143
plotLines(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *156*, *158*, *169*, *220*
plotMiss, 145
plotMiss(), *101*, *111*, *142*, *149*, *153*, *161*, *163*, *164*, *166*, *171*
plotMosaic, 146
plotMosaic(), *81*, *83*, *84*, *92*, *96*, *107*, *113*, *119*, *138*, *139*, *174*
plotPolar, 148
plotPolar(), *101*, *111*, *142*, *146*, *153*, *161*, *163*, *164*, *166*, *171*
plotProbDist, 150
plotProbDist(), *135*, *189*
plotPropCI, 152
plotPropCI(), *101*, *111*, *142*, *146*, *149*, *161*, *163*, *164*, *166*, *171*
plotQQ, 154
plotQQ(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *158*, *169*
plotRidge, 156
plotRidge(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *156*, *163*, *169*
plotSankey, 159
plotSankey(), *101*, *111*, *142*, *146*, *149*, *153*, *163*, *164*, *166*, *171*
plotTernary, 162
plotTernary(), *101*, *111*, *142*, *146*, *149*, *153*, *161*, *164*, *166*, *171*
plotTimeSeries, 163
plotTimeSeries(), *101*, *111*, *142*, *146*, *149*, *153*, *161*, *163*, *166*, *171*
plotTreemap, 164
plotTreemap(), *101*, *111*, *142*, *146*, *149*, *153*, *161*, *163*, *164*, *171*
plotViolin, 166
plotViolin(), *89*, *99*, *104*, *109*, *117*, *121*, *124*, *127*, *133*, *144*, *156*, *158*
plotWeb, 170
plotWeb(), *101*, *111*, *142*, *146*, *149*, *153*, *161*, *163*, *164*, *166*
plotXY, 172
plotXY(), *92*, *96*, *107*, *113*, *119*, *138*, *139*, *148*
points(), *45*, *46*
points.Lc *(*plot.Lc*)*, 85
points.LcList *(*plot.Lc*)*, 85
polarGrid, 175
polarGrid(), *27*, *188*
polToCart *(*coordinate-conversions*)*, 40
polygonX, 177
polygonX(), *9*, *20*, *23*, *27*, *43*, *182*, *186*, *187*
Prefix, 178
preview, 179
preview(), *12–14*, *44*, *47*, *65*, *180*, *226*
preview.html, 180
preview.html(), *12–14*, *44*, *47*, *65*, *179*, *225*, *226*
print(), *180*
print.Style *(*style*)*, 215
print.Unit, 181
print.Unit(), *39*, *58*, *59*, *218*, *228*

qqline(), *155*
quantile(), *155*

radToDeg *(*degree-radians-conversion*)*, 42
recordPlot(), *49*
rect(), *31*, *188*
regexpr(), *208*
regPolygon, 182
regPolygon(), *9*, *20*, *23*, *27*, *43*, *177*, *178*, *186*, *187*
resetTheme *(*theme*)*, 220
rgb(), *33*
rgbToCmy, 183
rgbToCmy(), *33*
rgbToCol, 183
rgbToCol(), *32*
rgbToHex, 184
rgbToHex(), *32*
rgbToLong, 185
rgbToLong(), *33*
ring, 186
ring(), *9*, *20*, *23*, *27*, *43*, *177*, *178*, *182*
rotate, 187
rotate(), *43*, *226*, *227*
rug(), *133*

scatter.smooth(), *71*, *191*
segments(), *220*
set.seed(), *49*
setBackCol, 188
setBackCol(), *27*, *176*
setTheme *(*theme*)*, 220
shade, 189

shade(), *135*, *150*, *151*
smooth.spline(), *71*
sphToCart *(*coordinate-conversions*)*, 40
splineCI, *69*, *71*, 190
splineX *(*splineCI*)*, 190
spreadOut, 192
spreadOut(), *6*, *16*, *19*, *66*, *72*, *74*, *130*
stamp, 193
stamp(), *22*, *26*, *31*, *46*, *49*, *76*, *81*, *84*, *87*, *89*, *94*, *101*, *106*, *108*, *110*, *112*, *118*, *121*, *123*, *129*, *133*, *135*, *139*, *141*, *144*, *146*, *149*, *151*, *153*, *158*, *160*, *162*, *164*, *165*, *168*, *171*, *174*, *220*, *222–224*
stats::cor(), *113*
stats::density(), *116*, *117*, *121*, *132*, *157*
stats::dist(), *200*, *201*
stats::lm(), *174*
stats::loess(), *174*
stats::model.frame(), *25*, *191*
stats::na.omit(), *94*
stats::prop.test(), *153*
stats::qqline, *156*
stats::qqnorm, *156*
stats::qqplot, *156*
stats::quantile(), *127*
stats::smooth.spline(), *190*, *191*
stats::symnum, *58*
strAbbr, 194
strAbbr(), *203*
strAlign, 195
strAlign(), *56*, *203*, *207*, *213*, *217*
strCap, 196
strCap(), *203*
strChop, 197
strChop(), *203*, *208*
strCountWords, 199
strCountWords(), *204*
strDist, 199
strDist(), *204*
strExtract, 201
strExtract(), *203*
strExtractBetween, 202
strExtractBetween(), *202*, *203*
strheight(), *6*, *192*, *193*
string-overview, *195–199*, *201–203*, 203, *205–213*, *215*
stringi::stri_count_words(), *199*
stringi::stri_extract_all_regex(), *215*
stringi::stri_extract_first_regex(), *201*, *202*
stringi::stri_length(), *195*
stringi::stri_match_first_regex(), *203*
stringi::stri_pad(), *196*
stringi::stri_reverse(), *209*
stringi::stri_split_boundaries(), *197*, *210*
stringi::stri_split_fixed(), *211*
stringi::stri_split_regex(), *211*
stringi::stri_sub(), *195*, *196*
stringi::stri_trans_totitle(), *197*
stringi::stri_trim_right(), *196*
strIsNumeric, 204
strIsNumeric(), *204*
strLeft *(*strLeftRight*)*, 205
strLeft(), *198*, *204*
strLeftRight, 205
strLen, 206
strLen(), *204*
strPad, 207
strPad(), *204*, *213*
strPos, 208
strPos(), *204*
strRev, 209
strRev(), *204*
strRight *(*strLeftRight*)*, 205
strRight(), *204*
strSpell, 210
strSpell(), *204*
strSplit, 211
strSplit(), *204*
strTrim, 212
strTrim(), *204*, *210*
strTrunc, 213
strTrunc(), *204*, *207*
strVal, 214
strVal(), *204*
strwidth(), *6*, *49*
style, 215
style(), *39*, *57–59*, *181*, *228*
styles *(*style*)*, 215
styles(), *222*
substitute(), *49*
substr(), *198*
svg(), *48*
symbols(), *107*

table(), *137*
tapply(), *10*, *60*, *123*
text(), *6*, *31*, *194*, *224*
textLegend, 219
textLegend(), *22*, *26*, *31*, *46*, *194*, *224*
theme, *58*, *70*, *76*, *91*, *94*, *99*, *103*, *110*, *113*, *116*, *120*, *126*, *127*, *132*, *133*, *135*, *137*, *138*, *140*, *142*, *144*, *145*, *147*, *150*, *152–154*, *156*, *157*, *159*, *167*, *171*, *173*, *174*, *194*, *218*, 220
title(), *164*, *224*
titleRect, *130*, 224
titleRect(), *22*, *26*, *31*, *46*, *129*, *130*, *194*, *220*
toHtmlTable, 225
toHtmlTable(), *12–14*, *44*, *47*, *65*, *179*, *180*
transformXY, 226
transformXY(), *187*
ts(), *164*

unit, 228
unit(), *39*, *58*, *59*, *181*, *218*
unit<- *(*unit*)*, 228
utils::adist(), *201*

xy.coords(), *187*, *227*