Chapter 5

The graphics package

---

graphics-package *The R Graphics Package*

Description

R functions for base graphics

Details

This package contains functions for 'base' graphics. Base graphics are traditional S-like graphics, as opposed to the more recent grid graphics.

For a complete list of functions with individual help pages, use `library(help = "graphics")`.

Author(s)

R Core Team and contributors worldwide

Maintainer: R Core Team <R-core@r-project.org>

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

---

abline *Add Straight Lines to a Plot*

Description

This function adds one or more straight lines through the current plot.

Usage

```
abline(a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
       coef = NULL, untf = FALSE, ...)
```

Arguments

a, b	the intercept and slope, single values.
untf	logical asking whether to *untransform*. See 'Details'.
h	the y-value(s) for horizontal line(s).
v	the x-value(s) for vertical line(s).
coef	a vector of length two giving the intercept and slope.
reg	an object with a coef method. See 'Details'.
...	graphical parameters such as col, lty and lwd (possibly as vectors: see 'Details') and xpd and the line characteristics lend, ljoin and lmitre.

Details

Typical usages are

```
abline(a, b, ...)
abline(h =, ...)
abline(v =, ...)
abline(coef =, ...)
abline(reg =, ...)
```

The first form specifies the line in intercept/slope form (alternatively a can be specified on its own and is taken to contain the slope and intercept in vector form).

The h= and v= forms draw horizontal and vertical lines at the specified coordinates.

The coef form specifies the line by a vector containing the slope and intercept.

reg is a regression object with a coef method. If this returns a vector of length 1 then the value is taken to be the slope of a line through the origin, otherwise, the first 2 values are taken to be the intercept and slope.

If untf is true, and one or both axes are log-transformed, then a curve is drawn corresponding to a line in original coordinates, otherwise a line is drawn in the transformed coordinate system. The h and v parameters always refer to original coordinates.

The graphical parameters col, lty and lwd can be specified; see par for details. For the h= and v= usages they can be vectors of length greater than one, recycled as necessary.

Specifying an xpd argument for clipping overrides the global par("xpd") setting used otherwise.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

lines and segments for connected and arbitrary lines given by their *endpoints*. par.

Examples

```
## Setup up coordinate system (with x == y aspect ratio):
plot(c(-2,3), c(-1,5), type = "n", xlab = "x", ylab = "y", asp = 1)
## the x- and y-axis, and an integer grid
abline(h = 0, v = 0, col = "gray60")
text(1,0, "abline( h = 0 )", col = "gray60", adj = c(0, -.1))
abline(h = -1:5, v = -2:3, col = "lightgray", lty = 3)
abline(a = 1, b = 2, col = 2)
text(1,3, "abline( 1, 2 )", col = 2, adj = c(-.1, -.1))

## Simple Regression Lines:
require(stats)
sale5 <- c(6, 4, 9, 7, 6, 12, 8, 10, 9, 13)
plot(sale5)
abline(lsfit(1:10, sale5))
abline(lsfit(1:10, sale5, intercept = FALSE), col = 4) # less fitting

z <- lm(dist ~ speed, data = cars)
plot(cars)
abline(z) # equivalent to abline(reg = z) or
abline(coef = coef(z))

## trivial intercept model
abline(mC <- lm(dist ~ 1, data = cars)) ## the same as
abline(a = coef(mC), b = 0, col = "blue")
```

---

arrows *Add Arrows to a Plot*

Description

Draw arrows between pairs of points.

Usage

```
arrows(x0, y0, x1 = x0, y1 = y0, length = 0.25, angle = 30,
       code = 2, col = par("fg"), lty = par("lty"),
       lwd = par("lwd"), ...)
```

Arguments

x0, y0	coordinates of points **from** which to draw.
x1, y1	coordinates of points **to** which to draw. At least one must the supplied
length	length of the edges of the arrow head (in inches).
angle	angle from the shaft of the arrow to the edge of the arrow head.
code	integer code, determining *kind* of arrows to be drawn.
col, lty, lwd	graphical parameters, possible vectors. NA values in col cause the arrow to be omitted.
...	graphical parameters such as xpd and the line characteristics lend, ljoin and lmitre: see par.

Details

For each `i`, an arrow is drawn between the point `(x0[i], y0[i])` and the point `(x1[i], y1[i])`. The coordinate vectors will be recycled to the length of the longest.

If `code = 1` an arrowhead is drawn at `(x0[i], y0[i])` and if `code = 2` an arrowhead is drawn at `(x1[i], y1[i])`. If `code = 3` a head is drawn at both ends of the arrow. Unless `length = 0`, when no head is drawn.

The graphical parameters `col`, `lty` and `lwd` can be vectors of length greater than one and will be recycled if necessary.

The direction of a zero-length arrow is indeterminate, and hence so is the direction of the arrowheads. To allow for rounding error, arrowheads are omitted (with a warning) on any arrow of length less than 1/1000 inch.

Note

The first four arguments in the comparable S function are named `x1`, `y1`, `x2`, `y2`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`segments` to draw segments.

Examples

```
x <- stats::runif(12); y <- stats::rnorm(12)
i <- order(x, y); x <- x[i]; y <- y[i]
plot(x,y, main = "arrows(.) and segments(.)")
## draw arrows from point to point :
s <- seq(length(x)-1)  # one shorter than data
arrows(x[s], y[s], x[s+1], y[s+1], col = 1:3)
s <- s[-length(s)]
segments(x[s], y[s], x[s+2], y[s+2], col = "pink")
```

---

assocplot *Association Plots*

Description

Produce a Cohen-Friendly association plot indicating deviations from independence of rows and columns in a 2-dimensional contingency table.

Usage

```
assocplot(x, col = c("black", "red"), space = 0.3,
          main = NULL, xlab = NULL, ylab = NULL)
```

Arguments

x	a two-dimensional contingency table in matrix form.
col	a character vector of length two giving the colors used for drawing positive and negative Pearson residuals, respectively.
space	the amount of space (as a fraction of the average rectangle width and height) left between each rectangle.
main	overall title for the plot.
xlab	a label for the x axis. Defaults to the name (if any) of the row dimension in x.
ylab	a label for the y axis. Defaults to the name (if any) of the column dimension in x.

Details

For a two-way contingency table, the signed contribution to Pearson's χ^2 for cell i, j is $d_{ij} = (f_{ij} - e_{ij})/\sqrt{e_{ij}}$, where f_{ij} and e_{ij} are the observed and expected counts corresponding to the cell. In the Cohen-Friendly association plot, each cell is represented by a rectangle that has (signed) height proportional to d_{ij} and width proportional to $\sqrt{e_{ij}}$, so that the area of the box is proportional to the difference in observed and expected frequencies. The rectangles in each row are positioned relative to a baseline indicating independence ($d_{ij} = 0$). If the observed frequency of a cell is greater than the expected one, the box rises above the baseline and is shaded in the color specified by the first element of col, which defaults to black; otherwise, the box falls below the baseline and is shaded in the color specified by the second element of col, which defaults to red.

A more flexible and extensible implementation of association plots written in the grid graphics system is provided in the function assoc in the contributed package **vcd** (Meyer, Zeileis and Hornik, 2006).

References

Cohen, A. (1980), On the graphical display of the significant components in a two-way contingency table. *Communications in Statistics—Theory and Methods*, **9**, 1025–1041. doi:10.1080/03610928008827940.

Friendly, M. (1992), Graphical methods for categorical data. *SAS User Group International Conference Proceedings*, **17**, 190–200. http://datavis.ca/papers/sugi/sugi17.pdf

Meyer, D., Zeileis, A., and Hornik, K. (2006) The strucplot Framework: Visualizing Multi-Way Contingency Tables with **vcd**. *Journal of Statistical Software*, **17(3)**, 1–48. doi:10.18637/jss.v017.i03.

See Also

mosaicplot, chisq.test.

Examples

```
## Aggregate over sex:
x <- marginSums(HairEyeColor, c(1, 2))
x
assocplot(x, main = "Relation between hair and eye color")
```

`Axis` *Generic Function to Add an Axis to a Plot*

Description

Generic function to add a suitable axis to the current plot.

Usage

```
Axis(x = NULL, at = NULL, ..., side, labels = NULL)
```

Arguments

`x`	an object which indicates the range over which an axis should be drawn
`at`	the points at which tick-marks are to be drawn.
`side`	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
`labels`	this can either be a logical value specifying whether (numerical) annotations are to be made at the tickmarks, or a character or expression vector of labels to be placed at the tick points. If this is specified as a character or expression vector, `at` should be supplied and they should be the same length.
`...`	arguments to be passed to methods and perhaps then to `axis`.

Details

This is a generic function. It works in a slightly non-standard way: if `x` is supplied and non-NULL it dispatches on `x`, otherwise if `at` is supplied and non-NULL it dispatches on `at`, and the default action is to call `axis`, omitting argument `x`.

The idea is that for plots for which either or both of the axes are numerical but with a special interpretation, the standard plotting functions (including `boxplot`, `contour`, `coplot`, `filled.contour`, `pairs`, `plot.default`, `rug` and `stripchart`) will set up user coordinates and `Axis` will be called to label them appropriately.

There are `"Date"` and `"POSIXt"` methods which can pass an argument `format` on to the appropriate `axis` method (see `axis.POSIXct`).

Value

The numeric locations on the axis scale at which tick marks were drawn when the plot was first drawn (see 'Details').

This function is usually invoked for its side effect, which is to add an axis to an already existing plot.

See Also

`axis` (which is eventually called from all `Axis()` methods) in package **graphics**.

`axis` *Add an Axis to a Plot*

Description

Adds an axis to the current plot, allowing the specification of the side, position, labels, and other options.

Usage

```
axis(side, at = NULL, labels = TRUE, tick = TRUE, line = NA,
     pos = NA, outer = FALSE, font = NA, lty = "solid",
     lwd = 1, lwd.ticks = lwd, col = NULL, col.ticks = NULL,
     hadj = NA, padj = NA, gap.axis = NA, ...)
```

Arguments

`side`	an integer specifying which side of the plot the axis is to be drawn on. The axis is placed as follows: 1=below, 2=left, 3=above and 4=right.
`at`	the points at which tick-marks are to be drawn. Non-finite (infinite, `NaN` or `NA`) values are omitted. By default (when `NULL`) tickmark locations are computed, see 'Details' below.
`labels`	this can either be a logical value specifying whether (numerical) annotations are to be made at the tickmarks, or a character or expression vector of labels to be placed at the tick points. (Other objects are coerced by `as.graphicsAnnot`.) If this is not logical, `at` should also be supplied and of the same length. If `labels` is of length zero after coercion, it has the same effect as supplying `TRUE`.
`tick`	a logical value specifying whether tickmarks and an axis line should be drawn.
`line`	the number of lines into the margin at which the axis line will be drawn, if not `NA`.
`pos`	the coordinate at which the axis line is to be drawn: if not `NA` this overrides the value of `line`.
`outer`	a logical value indicating whether the axis should be drawn in the outer plot margin, rather than the standard plot margin.
`font`	font for text. Defaults to `par("font")`.
`lty`	line type for both the axis line and the tick marks.
`lwd`, `lwd.ticks`	line widths for the axis line and the tick marks. Zero or negative values will suppress the line or ticks.
`col`, `col.ticks`	colors for the axis line and the tick marks respectively. `col = NULL` means to use `par("fg")`, possibly specified inline, and `col.ticks = NULL` means to use whatever color `col` resolved to.
`hadj`	adjustment (see `par("adj")`) for all labels *parallel* ('horizontal') to the reading direction. If this is not a finite value, the default is used (centring for strings parallel to the axis, justification of the end nearest the axis otherwise).
`padj`	adjustment for each tick label *perpendicular* to the reading direction. For labels parallel to the axes, `padj = 0` means left or bottom alignment, and `padj = 1` means right or top alignment (relative to the line). This can be a vector given a value for each string, and will be recycled as necessary.

If padj is not a finite value (the default), the value of par("las") determines the adjustment. For strings plotted perpendicular to the axis the default is to centre the string.

gap.axis an optional (typically non-negative) numeric factor to be multiplied with the size of an 'm' to determine the minimal gap between labels that are drawn, see 'Details'. The default, NA, corresponds to 1 for tick labels drawn *parallel* to the axis and 0.25 otherwise, i.e., the default is equivalent to

```
perpendicular <- function(side, las) {
  is.x <- (side %% 2 == 1) # is horizontal x-axis
  ( is.x && (las %in% 2:3)) ||
  (!is.x && (las %in% 1:2))
}
gap.axis <- if(perpendicular(side, las)) 0.25 else 1
```

gap.axis may typically be relevant when at = .. tick-mark positions are specified explicitly.

... other graphical parameters may also be passed as arguments to this function, particularly, cex.axis, col.axis and font.axis for axis annotation, i.e. tick labels, mgp and xaxp or yaxp for positioning, tck or tcl for tick mark length and direction, las for vertical/horizontal label orientation, or fg instead of col, and xpd for clipping. See par on these.

Parameters xaxt (sides 1 and 3) and yaxt (sides 2 and 4) control if the axis is plotted at all.

Note that lab will partial match to argument labels unless the latter is also supplied. (Since the default axes have already been set up by plot.window, lab will not be acted on by axis.)

Details

The axis line is drawn from the lowest to the highest value of at, but will be clipped at the plot region. By default, only ticks which are drawn from points within the plot region (up to a tolerance for rounding error) are plotted, but the ticks and their labels may well extend outside the plot region. Use xpd = TRUE or xpd = NA to allow axes to extend further.

When at = NULL, pretty tick mark locations are computed internally (the same way axTicks(side) would) from par("xaxp") or "yaxp" and par("xlog") (or "ylog"). Note that these locations may change if an on-screen plot is resized (for example, if the plot argument asp (see plot.window) is set.)

If labels is not specified, the numeric values supplied or calculated for at are converted to character strings as if they were a numeric vector printed by print.default(digits = 7).

The code tries hard not to draw overlapping tick labels, and so will omit labels where they would abut or overlap previously drawn labels. This can result in, for example, every other tick being labelled. The ticks are drawn left to right or bottom to top, and space at least the size of an 'm', multiplied by gap.axis, is left between labels. In previous R versions, this applied only for labels written *parallel* to the axis direction, hence not for e.g., las = 2. Using gap.axis = -1 restores that (buggy) previous behaviour (in the perpendicular case).

If either line or pos is set, they (rather than par("mgp")[3]) determine the position of the axis line and tick marks, and the tick labels are placed par("mgp")[2] further lines into (or towards for pos) the margin.

Several of the graphics parameters affect the way axes are drawn. The vertical (for sides 1 and 3) positions of the axis and the tick labels are controlled by mgp[2:3] and mex, the size and direction of

the ticks is controlled by `tck` and `tcl` and the appearance of the tick labels by `cex.axis`, `col.axis` and `font.axis` with orientation controlled by `las` (but not `srt`, unlike S which uses `srt` if `at` is supplied and `las` if it is not). Note that `adj` is not supported and labels are always centered. See `par` for details.

Value

The numeric locations on the axis scale at which tick marks were drawn when the plot was first drawn (see 'Details').

This function is usually invoked for its side effect, which is to add an axis to an already existing plot.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`Axis` for a generic interface.

`axTicks` returns the axis tick locations corresponding to `at = NULL`; `pretty` is more flexible for computing pretty tick coordinates and does *not* depend on (nor adapt to) the coordinate system in use.

Several graphics parameters affecting the appearance are documented in `par`.

Examples

```
require(stats) # for rnorm
plot(1:4, rnorm(4), axes = FALSE)
axis(1, 1:4, LETTERS[1:4])
axis(2)
box() #- to make it look "as usual"

plot(1:7, rnorm(7), main = "axis() examples",
     type = "s", xaxt = "n", frame.plot = FALSE, col = "red")
axis(1, 1:7, LETTERS[1:7], col.axis = "blue")
# unusual options:
axis(4, col = "violet", col.axis = "dark violet", lwd = 2)
axis(3, col = "gold", lty = 2, lwd = 0.5)

# one way to have a custom x axis
plot(1:10, xaxt = "n")
axis(1, xaxp = c(2, 9, 7))

## Changing default gap between labels:
plot(0:100, type="n", axes=FALSE, ann=FALSE)
title(quote("axis(1, .., gap.axis = f)," ~~ f >= 0))
axis(2, at = 5*(0:20), las = 1, gap.axis = 1/4)
gaps <- c(4, 2, 1, 1/2, 1/4, 0.1, 0)
chG <- paste0(ifelse(gaps == 1, "default:  ", ""),
              "gap.axis=", formatC(gaps))
jj <- seq_along(gaps)
linG <- -2.5*(jj-1)
for(j in jj) {
    isD <- gaps[j] == 1 # is default
```

```
    axis (1, at=5*(0:20), gap.axis = gaps[j], padj=-1, line = linG[j],
          col.axis = if(isD) "forest green" else 1, font.axis= 1+isD)
}
mtext(chG, side=1, padj=-1, line = linG -1/2, cex=3/4,
      col = ifelse(gaps == 1, "forest green", "blue3"))
## now shrink the window (in x- and y-direction) and observe the axis labels drawn
```

`axis.POSIXct` *Date and Date-time Plotting Functions*

Description

Add a date/time axis to the current plot of an object of class `"POSIXt"` or `"Date"`, respectively.

Usage

```
axis.POSIXct(side, x, at, format, labels = TRUE, ...)
axis.Date(side, x, at, format, labels = TRUE, ...)
```

Arguments

`side`	see `axis`.
`x`, `at`	optional date-time or Date objects, or other types of objects that can be converted appropriately.
`format`	an optional character string specifying the label format, see `strftime`.
`labels`	either a logical value specifying whether annotations are to be made at the tick-marks, or a character vector of labels to be placed at the tick points specified by `at`.
`...`	further arguments to be passed from or to other methods, typically graphical parameters.

Details

If `at` is unspecified, `axis.POSIXct` and `axis.Date` work quite hard (from R 4.3.0 via `pretty` for date-time classes) to choose suitable time units (years, months, days, hours, minutes, or seconds) and a sensible label `format` based on the axis range. `par("lab")` controls the approximate number of intervals.

If `at` is supplied it specifies the locations of the ticks and labels. If the label `format` is unspecified, a good guess is made by looking at the granularity of `at`. Printing of tick labels can be suppressed with `labels = FALSE`.

The date-times for a `"POSIXct"` input are interpreted in the time zone give by the `"tzone"` attribute if there is one, otherwise the current time zone.

The way the date-times are rendered (especially month names) is controlled by the locale setting of category `"LC_TIME"` (see `Sys.setlocale`).

Value

The locations on the axis scale at which tick marks were drawn.

See Also

DateTimeClasses, Dates for details of the classes.

Axis.

Examples

```
with(beaver1, {
    opar <- par(mfrow = c(3,1))
    time <- strptime(paste(1990, day, time %/% 100, time %% 100),
                     "%Y %j %H %M")
    plot(time, temp, type = "l") # axis at 6-hour intervals
    # request more ticks
    olab <- par(lab = c(10, 10, 7))
    plot(time, temp, type = "l")
    par(olab)
    # 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")
    par(opar) # reset changed par settings
})

plot(.leap.seconds, seq_along(.leap.seconds), type = "n", yaxt = "n",
     xlab = "leap seconds", ylab = "", bty = "n")
rug(.leap.seconds)
## or as dates
lps <- as.Date(.leap.seconds)
plot(lps, seq_along(.leap.seconds),
     type = "n", yaxt = "n", xlab = "leap seconds",
     ylab = "", bty = "n")
rug(lps)

## 100 random dates in a 10-week period
random.dates <- as.Date("2001/1/1") + 70*sort(stats::runif(100))
plot(random.dates, 1:100)
# or for a better axis labelling
plot(random.dates, 1:100, xaxt = "n")
axis.Date(1, at = seq(as.Date("2001/1/1"), max(random.dates)+6, "weeks"))
axis.Date(1, at = seq(as.Date("2001/1/1"), max(random.dates)+6, "days"),
     labels = FALSE, tcl = -0.2)

## axis.Date() with various data types:
x <- seq(as.Date("2022-01-20"), as.Date("2023-03-21"), by = "days")
plot(data.frame(x, y = 1), xaxt = "n")
legend("topleft", title = "input",
       legend = c("character", "Date", "POSIXct", "POSIXlt", "numeric"),
       fill = c("violet", "red", "orange", "coral1", "darkgreen"))
axis.Date(1)
axis.Date(3, at = "2022-04-01", col.axis = "violet")
axis.Date(3, at = as.Date("2022-07-01"), col.axis = "red")
axis.Date(3, at = as.POSIXct(as.Date("2022-10-01")), col.axis = "orange")
axis.Date(3, at = as.POSIXlt(as.Date("2023-01-01")), col.axis = "coral1")
axis.Date(3, at = as.integer(as.Date("2023-04-01")), col.axis = "darkgreen")
## automatically extends the format:
axis.Date(1, at = "2022-02-15", col.axis = "violet",
         col = "violet", tck = -0.05, mgp = c(3,2,0))
```

```
## axis.POSIXct() with various data types (2 minutes):
x <- as.POSIXct("2022-10-01") + c(0, 60, 120)
attributes(x)   # no timezone
plot(data.frame(x, y = 1), xaxt = "n")
legend("topleft", title = "input",
       legend = c("character", "Date", "POSIXct", "POSIXlt", "numeric"),
       fill = c("violet", "red", "orange", "coral1", "darkgreen"))
axis.POSIXct(1)
axis.POSIXct(3, at = "2022-10-01 00:01", col.axis = "violet")
axis.POSIXct(3, at = as.Date("2022-10-01"), col.axis = "red")
axis.POSIXct(3, at = as.POSIXct("2022-10-01 00:01:30"), col.axis = "orange")
axis.POSIXct(3, at = as.POSIXlt("2022-10-01 00:02"), col.axis = "coral1")
axis.POSIXct(3, at = as.numeric(as.POSIXct("2022-10-01 00:00:30")),
                col.axis = "darkgreen")
## automatically extends format (here: subseconds):
axis.POSIXct(3, at = as.numeric(as.POSIXct("2022-10-01 00:00:30")) + 0.25,
                col.axis = "forestgreen", col = "darkgreen", mgp = c(3,2,0))

## axis.POSIXct: 2 time zones
HST <- as.POSIXct("2022-10-01", tz = "HST") + c(0, 60, 60*60)
CET <- HST
attr(CET, "tzone") <- "CET"
plot(data.frame(HST, y = 1), xaxt = "n", xlab = "Hawaii Standard Time (HST)")
axis.POSIXct(1, HST)
axis.POSIXct(1, HST, at = "2022-10-01 00:10", col.axis = "violet")
axis.POSIXct(3, CET)
mtext(3, text = "Central European Time (CET)", line = 3)
axis.POSIXct(3, CET, at="2022-10-01 12:10", col.axis = "violet")
```

`axTicks` *Compute Axis Tickmark Locations*

Description

Compute pretty tickmark locations, the same way as R does internally. This is only non-trivial when **log** coordinates are active. By default, gives the `at` values which `axis(side)` would use.

Usage

```
axTicks(side, axp = NULL, usr = NULL, log = NULL, nintLog = NULL)
```

Arguments

`side`	integer in 1:4, as for `axis`.
`axp`	numeric vector of length three, defaulting to `par("xaxp")` or `par("yaxp")` depending on the `side` argument (`par("xaxp")` if `side` is 1 or 3, `par("yaxp")` if side is 2 or 4).
`usr`	numeric vector of length two giving user coordinate limits, defaulting to the relevant portion of `par("usr")` (`par("usr")[1:2]` or `par("usr")[3:4]` for `side` in (1,3) or (2,4) respectively).
`log`	logical indicating if log coordinates are active; defaults to `par("xlog")` or `par("ylog")` depending on `side`.

`nintLog` (only used when `log` is true): approximate (lower bound for the) number of tick intervals; defaults to `par("lab")[j]` where `j` is 1 or 2 depending on `side`. Set this to `Inf` if you want the same behavior as in earlier R versions (than 2.14.x).

Details

The `axp`, `usr`, and `log` arguments must be consistent as their default values (the `par(..)` results) are. If you specify all three (as non-NULL), the graphics environment is not used at all. Note that the meaning of `axp` differs significantly when `log` is `TRUE`; see the documentation on `par(xaxp = .)`.

`axTicks()` may be seen as an R implementation of the C function `CreateAtVector()` in '`..../src/main/plot.c`' which is called by `axis(side, *)` when no argument `at` is specified or directly by `axisTicks()` (in package **grDevices**).
The delicate case, `log = TRUE`, now makes use of `axisTicks` unless `nintLog = Inf` which exists for back compatibility.

Value

numeric vector of coordinate values at which axis tickmarks can be drawn. By default, when only the first argument is specified, these values should be identical to those that `axis(side)` would use or has used. Note that the values are decreasing when `usr` is ("reverse axis" case).

See Also

`axis`, `par`. `pretty` uses the same algorithm (but independently of the graphics environment) and has more options. However it is not available for `log = TRUE`.

`axisTicks()` (package **grDevices**).

Examples

```
 plot(1:7, 10*21:27)
 axTicks(1)
 axTicks(2)
 stopifnot(identical(axTicks(1), axTicks(3)),
           identical(axTicks(2), axTicks(4)))

## Show how axTicks() and axis() correspond :
op <- par(mfrow = c(3, 1))
for(x in 9999 * c(1, 2, 8)) {
    plot(x, 9, log = "x")
    cat(formatC(par("xaxp"), width = 5),";", T <- axTicks(1),"\n")
    rug(T, col =  adjustcolor("red", 0.5), lwd = 4)
}
par(op)

x <- 9.9*10^(-3:10)
plot(x, 1:14, log = "x")
axTicks(1) # now length 7
axTicks(1, nintLog = Inf) # rather too many

## An example using axTicks() without reference to an existing plot
## (copying R's internal procedures for setting axis ranges etc.),
## You do need to supply _all_ of axp, usr, log, nintLog
## standard logarithmic y axis labels
ylims <- c(0.2, 88)
```

```
get_axp <- function(x) 10^c(ceiling(x[1]), floor(x[2]))
## mimic par("yaxs") == "i"
usr.i <- log10(ylims)
(aT.i <- axTicks(side = 2, usr = usr.i,
                 axp = c(get_axp(usr.i), n = 3), log = TRUE, nintLog = 5))
## mimic (default) par("yaxs") == "r"
usr.r <- extendrange(r = log10(ylims), f = 0.04)
(aT.r <- axTicks(side = 2, usr = usr.r,
                 axp = c(get_axp(usr.r), 3), log = TRUE, nintLog = 5))

## Prove that we got it right :
plot(0:1, ylims, log = "y", yaxs = "i")
stopifnot(all.equal(aT.i, axTicks(side = 2)))

plot(0:1, ylims, log = "y", yaxs = "r")
stopifnot(all.equal(aT.r, axTicks(side = 2)))
```

---

barplot *Bar Plots*

Description

Creates a bar plot with vertical or horizontal bars.

Usage

```
barplot(height, ...)

## Default S3 method:
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
        add = FALSE, ann = !add && par("ann"), args.legend = NULL, ...)

## S3 method for class 'formula'
barplot(formula, data, subset, na.action,
        horiz = FALSE, xlab = NULL, ylab = NULL, ...)
```

Arguments

height	either a vector or matrix of values describing the bars which make up the plot. If height is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If height is a matrix and beside is FALSE then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked sub-bars making up the bar. If height is a matrix and beside is TRUE, then the values in each column are juxtaposed rather than stacked.

width	optional vector of bar widths. Re-cycled to length the number of bars drawn. Specifying a single value will have no visible effect unless xlim is specified.
space	the amount of space (as a fraction of the average bar width) left before each bar. May be given as a single number or one number per bar. If height is a matrix and beside is TRUE, space may be specified by two numbers, where the first is the space between bars in the same group, and the second the space between the groups. If not given explicitly, it defaults to c(0,1) if height is a matrix and beside is TRUE, and to 0.2 otherwise.
names.arg	a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the names attribute of height if this is a vector, or the column names if it is a matrix.
legend.text	a vector of text used to construct a legend for the plot, or a logical indicating whether a legend should be included. This is only useful when height is a matrix. In that case given legend labels should correspond to the rows of height; if legend.text is true, the row names of height will be used as labels if they are non-null.
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.
density	a vector giving the density of shading lines, in lines per inch, for the bars or bar components. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise), for the bars or bar components.
col	a vector of colors for the bars or bar components. By default, "grey" is used if height is a vector, and a gamma-corrected grey palette if height is a matrix; see grey.colors.
border	the color to be used for the border of the bars. Use border = NA to omit borders. If there are shading lines, border = TRUE means use the same colour for the border as for the shading lines.
main, sub	main title and subtitle for the plot.
xlab	a label for the x axis.
ylab	a label for the y axis.
xlim	limits for the x axis.
ylim	limits for the y axis.
xpd	logical. Should bars be allowed to go outside region?
log	string specifying if axis scales should be logarithmic; see plot.default.
axes	logical. If TRUE, a vertical (or horizontal, if horiz is true) axis is drawn.
axisnames	logical. If TRUE, and if there are names.arg (see above), the other axis is drawn (with lty = 0) and labeled.
cex.axis	expansion factor for numeric axis labels (see par('cex')).
cex.names	expansion factor for axis names (bar labels).
inside	logical. If TRUE, the lines which divide adjacent (non-stacked!) bars will be drawn. Only applies when space = 0 (which it partly is when beside = TRUE).
plot	logical. If FALSE, nothing is plotted.

axis.lty	the graphics parameter lty (see par('lty')) applied to the axis and tick marks of the categorical (default horizontal) axis. Note that by default the axis is suppressed.
offset	a vector indicating how much the bars should be shifted relative to the x axis.
add	logical specifying if bars should be added to an already existing plot; defaults to FALSE.
ann	logical specifying if the default annotation (main, sub, xlab, ylab) should appear on the plot, see title.
args.legend	list of additional arguments to pass to legend(); names of the list are used as argument names. Only used if legend.text is supplied.

formula	a formula where the y variables are numeric data to plot against the categorical x variables. The formula can have one of three forms:
	`y ~ x`
	`y ~ x1 + x2`
	`cbind(y1, y2) ~ x`
	(see the examples).

data	a data frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used.
na.action	a function which indicates what should happen when the data contain NA values. The default is to ignore missing values in the given variables.
...	arguments to be passed to/from other methods. For the default method these can include further arguments (such as axes, asp and main) and graphical parameters (see par) which are passed to plot.window(), title() and axis.

Value

A numeric vector (or matrix, when beside = TRUE), say mp, giving the coordinates of *all* the bar midpoints drawn, useful for adding to the graph.

If beside is true, use colMeans(mp) for the midpoints of each *group* of bars, see example.

Author(s)

R Core, with a contribution by Arni Magnusson.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot(..., type = "h"), dotchart; hist for bars of a *continuous* variable. mosaicplot(), more sophisticated to visualize *several* categorical variables.

Examples

```
# Formula method
barplot(GNP ~ Year, data = longley)
barplot(cbind(Employed, Unemployed) ~ Year, data = longley)

## 3rd form of formula - 2 categories :
op <- par(mfrow = 2:1, mgp = c(3,1,0)/2, mar = .1+c(3,3:1))
summary(d.Titanic <- as.data.frame(Titanic))
barplot(Freq ~ Class + Survived, data = d.Titanic,
        subset = Age == "Adult" & Sex == "Male",
     main = "barplot(Freq ~ Class + Survived, *)", ylab = "# {passengers}", legend.text = TRUE)
# Corresponding table :
(xt <- xtabs(Freq ~ Survived + Class + Sex, d.Titanic, subset = Age=="Adult"))
# Alternatively, a mosaic plot :
mosaicplot(xt[,,"Male"], main = "mosaicplot(Freq ~ Class + Survived, *)", color=TRUE)
par(op)


# Default method
require(grDevices) # for colours
tN <- table(Ni <- stats::rpois(100, lambda = 5))
r <- barplot(tN, col = rainbow(20))
#- type = "h" plotting *is* 'bar'plot
lines(r, tN, type = "h", col = "red", lwd = 2)

barplot(tN, space = 1.5, axisnames = FALSE,
        sub = "barplot(..., space= 1.5, axisnames = FALSE)")

barplot(VADeaths, plot = FALSE)
barplot(VADeaths, plot = FALSE, beside = TRUE)

mp <- barplot(VADeaths) # default
tot <- colMeans(VADeaths)
text(mp, tot + 3, format(tot), xpd = TRUE, col = "blue")
barplot(VADeaths, beside = TRUE,
        col = c("lightblue", "mistyrose", "lightcyan",
                "lavender", "cornsilk"),
        legend.text = rownames(VADeaths), ylim = c(0, 100))
title(main = "Death Rates in Virginia", font.main = 4)

hh <- t(VADeaths)[, 5:1]
mybarcol <- "gray20"
mp <- barplot(hh, beside = TRUE,
        col = c("lightblue", "mistyrose",
                "lightcyan", "lavender"),
        legend.text = colnames(VADeaths), ylim = c(0,100),
        main = "Death Rates in Virginia", font.main = 4,
        sub = "Faked upper 2*sigma error bars", col.sub = mybarcol,
        cex.names = 1.5)
segments(mp, hh, mp, hh + 2*sqrt(1000*hh/100), col = mybarcol, lwd = 1.5)
stopifnot(dim(mp) == dim(hh))  # corresponding matrices
mtext(side = 1, at = colMeans(mp), line = -2,
      text = paste("Mean", formatC(colMeans(hh))), col = "red")

# Bar shading example
barplot(VADeaths, angle = 15+10*1:5, density = 20, col = "black",
```

```
        legend.text = rownames(VADeaths))
title(main = list("Death Rates in Virginia", font = 4))

# Border color
barplot(VADeaths, border = "dark blue")

# Log scales (not much sense here)
barplot(tN, col = heat.colors(12), log = "y")
barplot(tN, col = gray.colors(20), log = "xy")

# Legend location
barplot(height = cbind(x = c(465, 91) / 465 * 100,
                       y = c(840, 200) / 840 * 100,
                       z = c(37, 17) / 37 * 100),
        beside = FALSE,
        width = c(465, 840, 37),
        col = c(1, 2),
        legend.text = c("A", "B"),
        args.legend = list(x = "topleft"))
```

---

box *Draw a Box around a Plot*

Description

This function draws a box around the current plot in the given color and line type. The `bty` parameter determines the type of box drawn. See `par` for details.

Usage

```
box(which = "plot", lty = "solid", ...)
```

Arguments

`which`	character, one of `"plot"`, `"figure"`, `"inner"` and `"outer"`.
`lty`	line type of the box.
`...`	further graphical parameters, such as `bty`, `col`, or `lwd`, see `par`. Note that `xpd` is not accepted as clipping is always to the device region.

Details

The choice of colour is complicated. If `col` was supplied and is not `NA`, it is used. Otherwise, if `fg` was supplied and is not `NA`, it is used. The final default is `par("col")`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`rect` for drawing of arbitrary rectangles.

Examples

```
plot(1:7, abs(stats::rnorm(7)), type = "h", axes = FALSE)
axis(1, at = 1:7, labels = letters[1:7])
box(lty = '1373', col = 'red')
```

---

boxplot *Box Plots*

Description

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

Usage

```
boxplot(x, ...)

## S3 method for class 'formula'
boxplot(formula, data = NULL, ..., subset, na.action = NULL,
        xlab = mklab(y_var = horizontal),
        ylab = mklab(y_var =!horizontal),
        add = FALSE, ann = !add, horizontal = FALSE,
        drop = FALSE, sep = ".", lex.order = FALSE)

## Default S3 method:
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
        notch = FALSE, outline = TRUE, names, plot = TRUE,
        border = par("fg"), col = "lightgray", log = "",
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
         ann = !add, horizontal = FALSE, add = FALSE, at = NULL)
```

Arguments

formula	a formula, such as y ~ grp, where y is a numeric vector of data values to be split into groups according to the grouping variable grp (usually a factor). Note that ~ g1 + g2 is equivalent to g1:g2.
data	a data.frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used for plotting.
na.action	a function which indicates what should happen when the data contain NAs. The default is to ignore missing values in either the response or the group.
xlab, ylab	x- and y-axis annotation, since R 3.6.0 with a non-empty default. Can be suppressed by ann=FALSE.
ann	logical indicating if axes should be annotated (by xlab and ylab).
drop, sep, lex.order	passed to split.default, see there.
x	for specifying data from which the boxplots are to be produced. Either a numeric vector, or a single list containing such vectors. Additional unnamed arguments specify further data as separate vectors (each corresponding to a component boxplot). NAs are allowed in the data.

`...`	For the `formula` method, named arguments to be passed to the default method. For the default method, unnamed arguments are additional data vectors (unless `x` is a list when they are ignored), and named arguments are arguments and graphical parameters to be passed to `bxp` in addition to the ones given by argument `pars` (and override those in `pars`). Note that `bxp` may or may not make use of graphical parameters it is passed: see its documentation.
`range`	this determines how far the plot whiskers extend out from the box. If `range` is positive, the whiskers extend to the most extreme data point which is no more than `range` times the interquartile range from the box. A value of zero causes the whiskers to extend to the data extremes.
`width`	a vector giving the relative widths of the boxes making up the plot.
`varwidth`	if `varwidth` is `TRUE`, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.
`notch`	if `notch` is `TRUE`, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap this is 'strong evidence' that the two medians differ (Chambers et al., 1983, p. 62). See `boxplot.stats` for the calculations used.
`outline`	if `outline` is not true, the outliers are not drawn (as points whereas S+ uses lines).
`names`	group labels which will be printed under each boxplot. Can be a character vector or an expression (see plotmath).
`boxwex`	a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower.
`staplewex`	staple line width expansion, proportional to box width.
`outwex`	outlier line width expansion, proportional to box width.
`plot`	if `TRUE` (the default) then a boxplot is produced. If not, the summaries which the boxplots are based on are returned.
`border`	an optional vector of colors for the outlines of the boxplots. The values in `border` are recycled if the length of `border` is less than the number of plots.
`col`	if `col` is non-null it is assumed to contain colors to be used to colour the bodies of the box plots. By default they are in the background colour.
`log`	character indicating if x or y or both coordinates should be plotted in log scale.
`pars`	a list of (potentially many) more graphical parameters, e.g., `boxwex` or `outpch`; these are passed to `bxp` (if `plot` is true); for details, see there.
`horizontal`	logical indicating if the boxplots should be horizontal; default `FALSE` means vertical boxes.
`add`	logical, if true *add* boxplot to current plot.
`at`	numeric vector giving the locations where the boxplots should be drawn, particularly when `add = TRUE`; defaults to `1:n` where `n` is the number of boxes.

Details

The generic function `boxplot` currently has a default method (`boxplot.default`) and a formula interface (`boxplot.formula`).

If multiple groups are supplied either as multiple arguments or via a formula, parallel boxplots will be plotted, in the order of the arguments or the order of the levels of the factor (see `factor`).

Missing values are ignored when forming boxplots.

Value

List with the following components:

stats	a matrix, each column contains the extreme of the lower whisker, the lower hinge, the median, the upper hinge and the extreme of the upper whisker for one group/plot. If all the inputs have the same class attribute, so will this component.
n	a vector with the number of (non-NA) observations in each group.
conf	a matrix where each column contains the lower and upper extremes of the notch.
out	the values of any data points which lie beyond the extremes of the whiskers.
group	a vector of the same length as out whose elements indicate to which group the outlier belongs.
names	a vector of names for the groups.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983). *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole.

Murrell, P. (2005). *R Graphics*. Chapman & Hall/CRC Press.

See also boxplot.stats.

See Also

boxplot.stats which does the computation, bxp for the plotting and more examples; and stripchart for an alternative (with small data sets).

Examples

```
## boxplot on a formula:
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")
# *add* notches (somewhat funny here <--> warning "notches .. outside hinges"):
boxplot(count ~ spray, data = InsectSprays,
        notch = TRUE, add = TRUE, col = "blue")

boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque",
        log = "y")
## horizontal=TRUE, switching  y <--> x :
boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque",
        log = "x", horizontal=TRUE)

rb <- boxplot(decrease ~ treatment, data = OrchardSprays, col = "bisque")
title("Comparing boxplot()s and non-robust mean +/- SD")
mn.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, mean)
sd.t <- tapply(OrchardSprays$decrease, OrchardSprays$treatment, sd)
xi <- 0.3 + seq(rb$n)
points(xi, mn.t, col = "orange", pch = 18)
arrows(xi, mn.t - sd.t, xi, mn.t + sd.t,
       code = 3, col = "pink", angle = 75, length = .1)

## boxplot on a matrix:
mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100),
```

```
             `5T` = rt(100, df = 5), Gam2 = rgamma(100, shape = 2))
boxplot(mat) # directly, calling boxplot.matrix()

## boxplot on a data frame:
df. <- as.data.frame(mat)
par(las = 1) # all axis labels horizontal
boxplot(df., main = "boxplot(*, horizontal = TRUE)", horizontal = TRUE)

## Using 'at = ' and adding boxplots -- example idea by Roger Bivand :
boxplot(len ~ dose, data = ToothGrowth,
        boxwex = 0.25, at = 1:3 - 0.2,
        subset = supp == "VC", col = "yellow",
        main = "Guinea Pigs' Tooth Growth",
        xlab = "Vitamin C dose mg",
        ylab = "tooth length",
        xlim = c(0.5, 3.5), ylim = c(0, 35), yaxs = "i")
boxplot(len ~ dose, data = ToothGrowth, add = TRUE,
        boxwex = 0.25, at = 1:3 + 0.2,
        subset = supp == "OJ", col = "orange")
legend(2, 9, c("Ascorbic acid", "Orange juice"),
       fill = c("yellow", "orange"))

## With less effort (slightly different) using factor *interaction*:
boxplot(len ~ dose:supp, data = ToothGrowth,
        boxwex = 0.5, col = c("orange", "yellow"),
        main = "Guinea Pigs' Tooth Growth",
        xlab = "Vitamin C dose mg", ylab = "tooth length",
        sep = ":", lex.order = TRUE, ylim = c(0, 35), yaxs = "i")

## more examples in  help(bxp)
```

`boxplot.matrix` *Draw a Boxplot for each Column (Row) of a Matrix*

Description

Interpreting the columns (or rows) of a matrix as different groups, draw a boxplot for each.

Usage

```
## S3 method for class 'matrix'
boxplot(x, use.cols = TRUE, ...)
```

Arguments

`x`	a numeric matrix.
`use.cols`	logical indicating if columns (by default) or rows (`use.cols = FALSE`) should be plotted.
`...`	Further arguments to `boxplot`.

Value

A list as for `boxplot`.

Author(s)

Martin Maechler, 1995, for S+, then R package **sfsmisc**.

See Also

`boxplot.default` which already works nowadays with data.frames; `boxplot.formula`, `plot.factor` which work with (the more general concept) of a grouping factor.

Examples

```
## Very similar to the example in ?boxplot
mat <- cbind(Uni05 = (1:100)/21, Norm = rnorm(100),
             T5 = rt(100, df = 5), Gam2 = rgamma(100, shape = 2))
boxplot(mat, main = "boxplot.matrix(...., main = ...)",
        notch = TRUE, col = 1:4)
```

---

bxp *Draw Box Plots from Summaries*

Description

bxp draws box plots based on the given summaries in z. It is usually called from within `boxplot`, but can be invoked directly.

Usage

```
bxp(z, notch = FALSE, width = NULL, varwidth = FALSE,
    outline = TRUE, notch.frac = 0.5, log = "",
    border = par("fg"), pars = NULL, frame.plot = axes,
    horizontal = FALSE, ann = TRUE,
    add = FALSE, at = NULL, show.names = NULL,
    ...)
```

Arguments

z	a list containing data summaries to be used in constructing the plots. These are usually the result of a call to `boxplot`, but can be generated in any fashion.
notch	if notch is TRUE, a notch is drawn in each side of the boxes. If the notches of two plots do not overlap then the medians are significantly different at the 5 percent level.
width	a vector giving the relative widths of the boxes making up the plot.
varwidth	if varwidth is TRUE, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.
outline	if outline is not true, the outliers are not drawn.
notch.frac	numeric in (0,1). When notch = TRUE, the fraction of the box width that the notches should use.
border	character or numeric (vector), the color of the box borders. Is recycled for multiple boxes. Is used as default for the boxcol, medcol, whiskcol, staplecol, and outcol options (see below).

`log` character, indicating if any axis should be drawn in logarithmic scale, as in `plot.default`.

`frame.plot` logical, indicating if a 'frame' (`box`) should be drawn; defaults to `TRUE`, unless `axes = FALSE` is specified.

`horizontal` logical indicating if the boxplots should be horizontal; default `FALSE` means vertical boxes.

`ann` a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.

`add` logical, if true *add* boxplot to current plot.

`at` numeric vector giving the locations where the boxplots should be drawn, particularly when `add = TRUE`; defaults to `1:n` where `n` is the number of boxes.

`show.names` Set to `TRUE` or `FALSE` to override the defaults on whether an x-axis label is printed for each group.

`pars, ...` graphical parameters (etc) can be passed as arguments to this function, either as a list (`pars`) or normally(`...`), see the following. (Those in `...` take precedence over those in `pars`.)

Currently, `yaxs` and `ylim` are used 'along the boxplot', i.e., vertically, when `horizontal` is false, and `xlim` horizontally. `xaxt`, `yaxt`, `las`, `cex.axis`, `gap.axis`, and `col.axis` are passed to `axis`, and `main`, `cex.main`, `col.main`, `sub`, `cex.sub`, `col.sub`, `xlab`, `ylab`, `cex.lab`, and `col.lab` are passed to `title`.

In addition, `axes` is accepted (see `plot.window`), with default `TRUE`.

The following arguments (or `pars` components) allow further customization of the boxplot graphics. Their defaults are typically determined from the non-prefixed version (e.g., `boxlty` from `lty`), either from the specified argument or `pars` component or the corresponding `par` one.

boxwex: a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower. The default depends on `at` and typically is 0.8.

staplewex, outwex: staple and outlier line width expansion, proportional to box width; both default to 0.5.

boxlty, boxlwd, boxcol, boxfill: box outline type, width, color, and fill color (which currently defaults to `col` and will in future default to `par("bg")`).

medlty, medlwd, medpch, medcex, medcol, medbg: median line type, line width, point character, point size expansion, color, and background color. The default `medpch = NA` suppresses the point, and `medlty = "blank"` does so for the line. Note that `medlwd` defaults to $3\times$ the default `lwd`.

whisklty, whisklwd, whiskcol: whisker line type (default: `"dashed"`), width, and color.

staplelty, staplelwd, staplecol: staple (= end of whisker) line type, width, and color.

outlty, outlwd, outpch, outcex, outcol, outbg: outlier line type, line width, point character, point size expansion, color, and background color. The default `outlty = "blank"` suppresses the lines and `outpch = NA` suppresses points.

Value

An invisible vector, actually identical to the `at` argument, with the coordinates ("x" if horizontal is false, "y" otherwise) of box centers, useful for adding to the plot.

Note

When add = FALSE, xlim now defaults to xlim = range(at, *) + c(-0.5, 0.5). It will usually be a good idea to specify xlim if the "x" axis has a log scale or width is far from uniform.

Author(s)

The R Core development team and Arni Magnusson (then at U Washington) who has provided most changes for the box*, med*, whisk*, staple*, and out* arguments.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```
require(stats)
set.seed(753)
(bx.p <- boxplot(split(rt(100, 4), gl(5, 20))))
op <- par(mfrow =  c(2, 2))
bxp(bx.p, xaxt = "n")
bxp(bx.p, notch = TRUE, axes = FALSE, pch = 4, boxfill = 1:5)
bxp(bx.p, notch = TRUE, boxfill = "lightblue", frame.plot = FALSE,
    outline = FALSE, main = "bxp(*, frame.plot= FALSE, outline= FALSE)")
bxp(bx.p, notch = TRUE, boxfill = "lightblue", border = 2:6,
    ylim = c(-4,4), pch = 22, bg = "green", log = "x",
    main = "... log = 'x', ylim = *")
par(op)
op <- par(mfrow = c(1, 2))

## single group -- no label
boxplot (weight ~ group, data = PlantGrowth, subset = group == "ctrl")
## with label
bx <- boxplot(weight ~ group, data = PlantGrowth,
              subset = group == "ctrl", plot = FALSE)
bxp(bx, show.names=TRUE)
par(op)

## passing gap.axis=* to axis(), PR#18109:
boxplot(matrix(100*rnorm(1e3), 50, 20),
        cex.axis = 1.5, gap.axis = -1)# showing *all* labels

z <- split(rnorm(1000), rpois(1000, 2.2))
boxplot(z, whisklty = 3, main = "boxplot(z, whisklty = 3)")

## Colour support similar to plot.default:
op <- par(mfrow = 1:2, bg = "light gray", fg = "midnight blue")
boxplot(z,   col.axis = "skyblue3", main = "boxplot(*, col.axis=..,main=..)")
plot(z[[1]], col.axis = "skyblue3", main =    "plot(*, col.axis=..,main=..)")
mtext("par(bg=\"light gray\", fg=\"midnight blue\")",
      outer = TRUE, line = -1.2)
par(op)

## Mimic S-Plus:
splus <- list(boxwex = 0.4, staplewex = 1, outwex = 1, boxfill = "grey40",
              medlwd = 3, medcol = "white", whisklty = 3, outlty = 1, outpch = NA)
```

```
boxplot(z, pars = splus)
## Recycled and "sweeping" parameters
op <- par(mfrow = c(1,2))
 boxplot(z, border = 1:5, lty = 3, medlty = 1, medlwd = 2.5)
 boxplot(z, boxfill = 1:3, pch = 1:5, lwd = 1.5, medcol = "white")
par(op)
## too many possibilities
boxplot(z, boxfill = "light gray", outpch = 21:25, outlty = 2,
        bg = "pink", lwd = 2,
        medcol = "dark blue", medcex = 2, medpch = 20)
```

---

cdplot *Conditional Density Plots*

Description

Computes and plots conditional densities describing how the conditional distribution of a categorical variable y changes over a numerical variable x.

Usage

```
cdplot(x, ...)

## Default S3 method:
cdplot(x, y,
  plot = TRUE, tol.ylab = 0.05, ylevels = NULL,
  bw = "nrd0", n = 512, from = NULL, to = NULL,
  col = NULL, border = 1, main = "", xlab = NULL, ylab = NULL,
  yaxlabels = NULL, xlim = NULL, ylim = c(0, 1), weights = NULL, ...)

## S3 method for class 'formula'
cdplot(formula, data = list(),
  plot = TRUE, tol.ylab = 0.05, ylevels = NULL,
  bw = "nrd0", n = 512, from = NULL, to = NULL,
  col = NULL, border = 1, main = "", xlab = NULL, ylab = NULL,
  yaxlabels = NULL, xlim = NULL, ylim = c(0, 1), ...,
  subset = NULL, weights = NULL)
```

Arguments

x	an object, the default method expects a single numerical variable (or an object coercible to this).
y	a "factor" interpreted to be the dependent variable
formula	a "formula" of type y ~ x with a single dependent "factor" and a single numerical explanatory variable.
data	an optional data frame.
plot	logical. Should the computed conditional densities be plotted?
tol.ylab	convenience tolerance parameter for y-axis annotation. If the distance between two labels drops under this threshold, they are plotted equidistantly.

ylevels	a character or numeric vector specifying in which order the levels of the dependent variable should be plotted.
bw, n, from, to, ...	arguments passed to density
col	a vector of fill colors of the same length as levels(y). The default is to call gray.colors.
border	border color of shaded polygons.
main, xlab, ylab	character strings for annotation
yaxlabels	character vector for annotation of y axis, defaults to levels(y).
xlim, ylim	the range of x and y values with sensible defaults.
subset	an optional vector specifying a subset of observations to be used for plotting.
weights	numeric. A vector of frequency weights for each observation in the data. If NULL all weights are implicitly assumed to be 1.

Details

cdplot computes the conditional densities of x given the levels of y weighted by the marginal distribution of y. The densities are derived cumulatively over the levels of y.

This visualization technique is similar to spinograms (see spineplot) and plots $P(y|x)$ against x. The conditional probabilities are not derived by discretization (as in the spinogram), but using a smoothing approach via density.

Note, that the estimates of the conditional densities are more reliable for high-density regions of x. Conversely, the are less reliable in regions with only few x observations.

Value

The conditional density functions (cumulative over the levels of y) are returned invisibly.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Hofmann, H., Theus, M. (2005), *Interactive graphics for visualizing conditional distributions*, Unpublished Manuscript.

See Also

spineplot, density

Examples

```
## NASA space shuttle o-ring failures
fail <- factor(c(2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1,
                 1, 2, 1, 1, 1, 1, 1),
               levels = 1:2, labels = c("no", "yes"))
temperature <- c(53, 57, 58, 63, 66, 67, 67, 67, 68, 69, 70, 70,
                 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 81)

## CD plot
```

```
cdplot(fail ~ temperature)
cdplot(fail ~ temperature, bw = 2)
cdplot(fail ~ temperature, bw = "SJ")

## compare with spinogram
(spineplot(fail ~ temperature, breaks = 3))

## highlighting for failures
cdplot(fail ~ temperature, ylevels = 2:1)

## scatter plot with conditional density
cdens <- cdplot(fail ~ temperature, plot = FALSE)
plot(I(as.numeric(fail) - 1) ~ jitter(temperature, factor = 2),
     xlab = "Temperature", ylab = "Conditional failure probability")
lines(53:81, 1 - cdens[[1]](53:81), col = 2)
```

---

clip *Set Clipping Region*

Description

Set clipping region in user coordinates

Usage

```
clip(x1, x2, y1, y2)
```

Arguments

x1, x2, y1, y2	user coordinates of clipping rectangle

Details

How the clipping rectangle is set depends on the setting of `par("xpd")`: this function changes the current setting until the next high-level plotting command resets it.

Clipping of lines, rectangles and polygons is done in the graphics engine, but clipping of text is if possible done in the device, so the effect of clipping text is device-dependent (and may result in text not wholly within the clipping region being omitted entirely).

Exactly when the clipping region will be reset can be hard to predict. `plot.new` always resets it. Functions such as `lines` and `text` only reset it if `par("xpd")` has been changed. However, functions such as `box`, `mtext`, `title` and `plot.dendrogram` can manipulate the xpd setting.

See Also

`par`

Examples

```
x <- rnorm(1000)
hist(x, xlim = c(-4,4))
usr <- par("usr")
clip(usr[1], -2, usr[3], usr[4])
hist(x, col = 'red', add = TRUE)
clip(2, usr[2], usr[3], usr[4])
hist(x, col = 'blue', add = TRUE)
do.call("clip", as.list(usr))  # reset to plot region
```

`contour` *Display Contours*

Description

Create a contour plot, or add contour lines to an existing plot.

Usage

```
contour(x, ...)

## Default S3 method:
contour(x = seq(0, 1, length.out = nrow(z)),
        y = seq(0, 1, length.out = ncol(z)),
        z,
        nlevels = 10, levels = pretty(zlim, nlevels),
        labels = NULL,
        xlim = range(x, finite = TRUE),
        ylim = range(y, finite = TRUE),
        zlim = range(z, finite = TRUE),
        labcex = 0.6, drawlabels = TRUE, method = "flattest",
        vfont, axes = TRUE, frame.plot = axes,
        col = par("fg"), lty = par("lty"), lwd = par("lwd"),
        add = FALSE, ...)
```

Arguments

x, y	locations of grid lines at which the values in z are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If x is a `list`, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z.
z	a matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience.
nlevels	number of contour levels desired **iff** `levels` is not supplied.
levels	numeric vector of levels at which to draw contour lines.
labels	a vector giving the labels for the contour lines. If NULL then the levels are used as labels, otherwise this is coerced by `as.character`.
labcex	cex for contour labelling. This is an absolute size, not a multiple of `par("cex")`.
drawlabels	logical. Contours are labelled if TRUE.

method	character string specifying where the labels will be located. Possible values are "simple", "edge" and "flattest" (the default). See the 'Details' section.
vfont	if NULL, the current font family and face are used for the contour labels. If a character vector of length 2 then Hershey vector fonts are used for the contour labels. The first element of the vector selects a typeface and the second element selects a font index (see text for more information). The default is NULL on graphics devices with high-quality rotation of text and c("sans serif", "plain") otherwise.
xlim, ylim, zlim	x-, y- and z-limits for the plot.
axes, frame.plot	logical indicating whether axes or a box should be drawn, see plot.default.
col	colour(s) for the lines drawn.
lty	line type(s) for the lines drawn.
lwd	line width(s) for the lines drawn.
add	logical. If TRUE, add to a current plot.
...	additional arguments to plot.window, title, Axis and box, typically graphical parameters such as cex.axis.

Details

contour is a generic function with only a default method in base R.

The methods for positioning the labels on contours are "simple" (draw at the edge of the plot, overlaying the contour line), "edge" (draw at the edge of the plot, embedded in the contour line, with no labels overlapping) and "flattest" (draw on the flattest section of the contour, embedded in the contour line, with no labels overlapping). The second and third may not draw a label on every contour line.

For information about vector fonts, see the help for text and Hershey.

Notice that contour interprets the z matrix as a table of f(x[i], y[j]) values, so that the x axis corresponds to row number and the y axis to column number, with column 1 at the bottom, i.e. a 90 degree counter-clockwise rotation of the conventional textual layout.

Vector (of length > 1) col, lty, and lwd are applied along levels and recycled, see the Examples.

Alternatively, use contourplot from the **lattice** package where the formula notation allows to use vectors x, y, and z of the same length.

There is limited control over the axes and frame as arguments col, lwd and lty refer to the contour lines (rather than being general graphical parameters). For more control, add contours to a plot, or add axes and frame to a contour plot.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

options("max.contour.segments") for the maximal complexity of a single contour line.

contourLines, filled.contour for color-filled contours, contourplot (and levelplot) from package **lattice**. Further, image and the graphics demo which can be invoked as demo(graphics).

Examples

```
require(grDevices) # for colours
x <- -6:16
op <- par(mfrow = c(2, 2))
contour(outer(x, x), method = "edge", vfont = c("sans serif", "plain"))
z <- outer(x, sqrt(abs(x)), FUN = `/`)
image(x, x, z)
contour(x, x, z, col = "pink", add = TRUE, method = "edge",
        vfont = c("sans serif", "plain"))
contour(x, x, z, ylim = c(1, 6), method = "simple", labcex = 1,
        xlab = quote(x[1]), ylab = quote(x[2]))
contour(x, x, z, ylim = c(-6, 6), nlevels = 20, lty = 2, method = "simple",
        main = "20 levels; \"simple\" labelling method")
par(op)

## Passing multiple colours / lty / lwd :
op <- par(mfrow = c(1, 2))
z <- outer(-9:25, -9:25)
## Using default levels <- pretty(range(z, finite = TRUE), 10),
##  the first and last of which typically are *not* drawn:
(levs <- pretty(z, n=10)) # -300 -200 ... 600 700
contour(z, col = 1:4)
## Set levels explicitly; show that 'lwd' and 'lty' are recycled as well:
contour(z, levels=levs[-c(1,length(levs))], col = 1:5, lwd = 1:3 *1.5, lty = 1:3)
par(op)

## Persian Rug Art:
x <- y <- seq(-4*pi, 4*pi, length.out = 27)
r <- sqrt(outer(x^2, y^2, `+`))
opar <- par(mfrow = c(2, 2), mar = rep(0, 4))
for(f in pi^(0:3))
  contour(cos(r^2)*exp(-r/f),
          drawlabels = FALSE, axes = FALSE, frame.plot = TRUE)

rx <- range(x <- 10*1:nrow(volcano))
ry <- range(y <- 10*1:ncol(volcano))
ry <- ry + c(-1, 1) * (diff(rx) - diff(ry))/2
tcol <- terrain.colors(12)
par(opar); opar <- par(pty = "s", bg = "lightcyan")
plot(x = 0, y = 0, type = "n", xlim = rx, ylim = ry, xlab = "", ylab = "")
u <- par("usr")
rect(u[1], u[3], u[2], u[4], col = tcol[8], border = "red")
contour(x, y, volcano, col = tcol[2], lty = "solid", add = TRUE,
        vfont = c("sans serif", "plain"))
title("A Topographic Map of Maunga Whau", font = 4)
abline(h = 200*0:4, v = 200*0:4, col = "lightgray", lty = 2, lwd = 0.1)

## contourLines produces the same contour lines as contour
plot(x = 0, y = 0, type = "n", xlim = rx, ylim = ry, xlab = "", ylab = "")
u <- par("usr")
rect(u[1], u[3], u[2], u[4], col = tcol[8], border = "red")
contour(x, y, volcano, col = tcol[1], lty = "solid", add = TRUE,
        vfont = c("sans serif", "plain"))
line.list <- contourLines(x, y, volcano)
invisible(lapply(line.list, lines, lwd=3, col=adjustcolor(2, .3)))
par(opar)
```

convertXY *Convert between Graphics Coordinate Systems*

Description

Convert between graphics coordinate systems.

Usage

```
grconvertX(x, from = "user", to = "user")
grconvertY(y, from = "user", to = "user")
```

Arguments

x, y	numeric vector of coordinates.
from, to	character strings giving the coordinate systems to convert between.

Details

The coordinate systems are

"user" user coordinates.

"inches" inches.

"device" the device coordinate system.

"ndc" normalized device coordinates.

"nfc" normalized figure coordinates.

"npc" normalized plot coordinates.

"nic" normalized inner region coordinates. (The 'inner region' is that inside the outer margins.)

"lines" lines of margin (based on mex).

"chars" lines of text (based on cex).

(These names can be partially matched.) For the 'normalized' coordinate systems the lower left has value 0 and the top right value 1.

Device coordinates are those in which the device works: they are usually in pixels where that makes sense and in big points (1/72 inch) otherwise (e.g., pdf and postscript).

Value

A numeric vector of the same length as the input.

Examples

```
op <- par(omd=c(0.1, 0.9, 0.1, 0.9), mfrow = c(1, 2))
plot(1:4)
for(tp in c("in", "dev", "ndc", "nfc", "npc", "nic", "lines", "chars"))
    print(grconvertX(c(1.0, 4.0), "user", tp))
par(op)
```

coplot *Conditioning Plots*

Description

This function produces two variants of the **co**nditioning plots discussed in the reference below.

Usage

```
coplot(formula, data, given.values, panel = points, rows, columns,
       show.given = TRUE, col = par("fg"), pch = par("pch"),
       bar.bg = c(num = gray(0.8), fac = gray(0.95)),
       xlab = c(x.name, paste("Given :", a.name)),
       ylab = c(y.name, paste("Given :", b.name)),
       subscripts = FALSE,
       axlabels = function(f) abbreviate(levels(f)),
       number = 6, overlap = 0.5, xlim, ylim, ...)
co.intervals(x, number = 6, overlap = 0.5)
```

Arguments

formula	a formula describing the form of conditioning plot. A formula of the form `y ~ x | a` indicates that plots of `y` versus `x` should be produced conditional on the variable `a`. A formula of the form `y ~ x | a * b` indicates that plots of `y` versus `x` should be produced conditional on the two variables `a` and `b`. All three or four variables may be either numeric or factors. When `x` or `y` are factors, the result is almost as if `as.numeric()` was applied, whereas for factor `a` or `b`, the conditioning (and its graphics if `show.given` is true) are adapted.
data	a data frame containing values for any variables in the formula. By default the environment where `coplot` was called from is used.
given.values	a value or list of two values which determine how the conditioning on `a` and `b` is to take place. When there is no `b` (i.e., conditioning only on `a`), usually this is a matrix with two columns each row of which gives an interval, to be conditioned on, but is can also be a single vector of numbers or a set of factor levels (if the variable being conditioned on is a factor). In this case (no `b`), the result of `co.intervals` can be used directly as `given.values` argument.
panel	a `function(x, y, col, pch, ...)` which gives the action to be carried out in each panel of the display. The default is `points`.
rows	the panels of the plot are laid out in a `rows` by `columns` array. `rows` gives the number of rows in the array.
columns	the number of columns in the panel layout array.
show.given	logical (possibly of length 2 for 2 conditioning variables): should conditioning plots be shown for the corresponding conditioning variables (default `TRUE`).
col	a vector of colors to be used to plot the points. If too short, the values are recycled.
pch	a vector of plotting symbols or characters. If too short, the values are recycled.

bar.bg	a named vector with components "num" and "fac" giving the background colors for the (shingle) bars, for **num**eric and **fac**tor conditioning variables respectively.
xlab	character; labels to use for the x axis and the first conditioning variable. If only one label is given, it is used for the x axis and the default label is used for the conditioning variable.
ylab	character; labels to use for the y axis and any second conditioning variable.
subscripts	logical: if true the panel function is given an additional (third) argument subscripts giving the subscripts of the data passed to that panel.
axlabels	function for creating axis (tick) labels when x or y are factors.
number	integer; the number of conditioning intervals, for a and b, possibly of length 2. It is only used if the corresponding conditioning variable is not a factor.
overlap	numeric < 1; the fraction of overlap of the conditioning variables, possibly of length 2 for x and y direction. When overlap < 0, there will be *gaps* between the data slices.
xlim	the range for the x axis.
ylim	the range for the y axis.
...	additional arguments to the panel function.
x	a numeric vector.

Details

In the case of a single conditioning variable a, when both rows and columns are unspecified, a 'close to square' layout is chosen with columns >= rows.

In the case of multiple rows, the *order* of the panel plots is from the bottom and from the left (corresponding to increasing a, typically).

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus plot and boxplot are not panel functions.

The rendering of arguments xlab and ylab is not controlled by par arguments cex.lab and font.lab even though they are plotted by mtext rather than title.

Value

co.intervals(., number, .) returns a (number $\times$ 2) matrix, say ci, where ci[k,] is the range of x values for the k-th interval.

References

Chambers, J. M. (1992) *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Cleveland, W. S. (1993) *Visualizing Data.* New Jersey: Summit Press.

See Also

pairs, panel.smooth, points.

Examples

```
## Tonga Trench Earthquakes
coplot(lat ~ long | depth, data = quakes)
given.depth <- co.intervals(quakes$depth, number = 4, overlap = .1)
coplot(lat ~ long | depth, data = quakes, given.values = given.depth, rows = 1)

## Conditioning on 2 variables:
ll.dm <- lat ~ long | depth * mag
coplot(ll.dm, data = quakes)
coplot(ll.dm, data = quakes, number = c(4, 7), show.given = c(TRUE, FALSE))
coplot(ll.dm, data = quakes, number = c(3, 7),
       overlap = c(-.5, .1)) # negative overlap DROPS values

## given two factors
Index <- seq_len(nrow(warpbreaks)) # to get nicer default labels
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       show.given = 0:1)
coplot(breaks ~ Index | wool * tension, data = warpbreaks,
       col = "red", bg = "pink", pch = 21,
       bar.bg = c(fac = "light blue"))

## Example with empty panels:
with(data.frame(state.x77), {
coplot(Life.Exp ~ Income | Illiteracy * state.region, number = 3,
       panel = function(x, y, ...) panel.smooth(x, y, span = .8, ...))
## y ~ factor -- not really sensible, but 'show off':
coplot(Life.Exp ~ state.region | Income * state.division,
       panel = panel.smooth)
})
```

---

curve *Draw Function Plots*

Description

Draws a curve corresponding to a function over the interval `[from, to]`. `curve` can plot also an expression in the variable `xname`, default '`x`'.

Usage

```
curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,
      type = "l", xname = "x", xlab = xname, ylab = NULL,
      log = NULL, xlim = NULL, ...)

## S3 method for class 'function'
plot(x, y = 0, to = 1, from = y, xlim = NULL, ylab = NULL, ...)
```

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`.
`x`	a 'vectorizing' numeric R function.
`y`	alias for `from` for compatibility with `plot`

from, to	the range over which the function will be plotted.
n	integer; the number of x values at which to evaluate.
add	logical; if TRUE add to an already existing plot; if NA start a new plot taking the defaults for the limits and log-scaling of the x-axis from the previous plot. Taken as FALSE (with a warning if a different value is supplied) if no graphics device is open.
xlim	NULL or a numeric vector of length 2; if non-NULL it provides the defaults for c(from, to) and, unless add = TRUE, selects the x-limits of the plot – see plot.window.
type	plot type: see plot.default.
xname	character string giving the name to be used for the x axis.
xlab, ylab, log, ...	labels and graphical parameters can also be specified as arguments. See 'Details' for the interpretation of the default for log. For the "function" method of plot, ... can include any of the other arguments of curve, except expr.

Details

The function or expression expr (for curve) or function x (for plot) is evaluated at n points equally spaced over the range [from, to]. The points determined in this way are then plotted.

If either from or to is NULL, it defaults to the corresponding element of xlim if that is not NULL.

What happens when neither from/to nor xlim specifies both x-limits is a complex story. For plot(<function>) and for curve(add = FALSE) the defaults are $(0, 1)$. For curve(add = NA) and curve(add = TRUE) the defaults are taken from the x-limits used for the previous plot. (This differs from versions of R prior to 2.14.0.)

The value of log is used both to specify the plot axes (unless add = TRUE) and how 'equally spaced' is interpreted: if the x component indicates log-scaling, the points at which the expression or function is plotted are equally spaced on log scale.

The default value of log is taken from the current plot when add = TRUE, whereas if add = NA the x component is taken from the existing plot (if any) and the y component defaults to linear. For add = FALSE the default is ""

This used to be a quick hack which now seems to serve a useful purpose, but can give bad results for functions which are not smooth.

For expensive-to-compute expressions, you should use smarter tools.

The way curve handles expr has caused confusion. It first looks to see if expr is a name (also known as a symbol), in which case it is taken to be the name of a function, and expr is replaced by a call to expr with a single argument with name given by xname. Otherwise it checks that expr is either a call or an expression, and that it contains a reference to the variable given by xname (using all.vars): anything else is an error. Then expr is evaluated in an environment which supplies a vector of name given by xname of length n, and should evaluate to an object of length n. Note that this means that curve(x, ...) is taken as a request to plot a function named x (and it is used as such in the function method for plot).

The plot method can be called directly as plot.function.

Value

A list with components x and y of the points that were drawn is returned invisibly.

Warning

For historical reasons, add is allowed as an argument to the "function" method of plot, but its behaviour may surprise you. It is recommended to use add only with curve.

See Also

splinefun for spline interpolation, lines.

Examples

```
plot(qnorm) # default range c(0, 1) is appropriate here,
            # but end values are -/+Inf and so are omitted.
plot(qlogis, main = "The Inverse Logit : qlogis()")
abline(h = 0, v = 0:2/2, lty = 3, col = "gray")

curve(sin, -2*pi, 2*pi, xname = "t")
curve(tan, xname = "t", add = NA,
      main = "curve(tan)  --> same x-scale as previous plot")

op <- par(mfrow = c(2, 2))
curve(x^3 - 3*x, -2, 2)
curve(x^2 - 2, add = TRUE, col = "violet")

## simple and advanced versions, quite similar:
plot(cos, -pi,  3*pi)
curve(cos, xlim = c(-pi, 3*pi), n = 1001, col = "blue", add = TRUE)

chippy <- function(x) sin(cos(x)*exp(-x/2))
curve(chippy, -8, 7, n = 2001)
plot (chippy, -8, -5)

for(ll in c("", "x", "y", "xy"))
   curve(log(1+x), 1, 100, log = ll, sub = paste0("log = '", ll, "'"))
par(op)
```

---

dotchart *Cleveland's Dot Plots*

Description

Draw a Cleveland dot plot.

Usage

```
dotchart(x, labels = NULL, groups = NULL, gdata = NULL, offset = 1/8,
         ann = par("ann"), xaxt = par("xaxt"), frame.plot = TRUE, log = "",
         cex = par("cex"), pt.cex = cex,
         pch = 21, gpch = 21, bg = par("bg"),
         color = par("fg"), gcolor = par("fg"), lcolor = "gray",
         xlim = range(x[is.finite(x)]),
         main = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

x	either a vector or matrix of numeric values (NAs are allowed). If x is a matrix the overall plot consists of juxtaposed dotplots for each row. Inputs which satisfy is.numeric(x) but not is.vector(x) || is.matrix(x) are coerced by as.numeric, with a warning.
labels	a vector of labels for each point. For vectors the default is to use names(x) and for matrices the row labels dimnames(x)[[1]].
groups	an optional factor indicating how the elements of x are grouped. If x is a matrix, groups will default to the columns of x.
gdata	data values for the groups. This is typically a summary such as the median or mean of each group.
offset	offset in inches of ylab and labels.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
xaxt	a string indicating the x-axis style; use "n" to suppress and see also par("xaxt").
frame.plot	a logical indicating whether a box should be drawn around the plot.
log	a character string indicating if one or the other axis should be logarithmic, see plot.default.
cex	the character size to be used. Setting cex to a value smaller than one can be a useful way of avoiding label overlap. Unlike many other graphics functions, this sets the actual size, not a multiple of par("cex").
pt.cex	the cex to be applied to plotting symbols. This behaves like cex in plot().
pch	the plotting character or symbol to be used.
gpch	the plotting character or symbol to be used for group values.
bg	the background color of plotting characters or symbols to be used; use par(bg= *) to set the background color of the whole plot.
color	the color(s) to be used for points and labels.
gcolor	the single color to be used for group labels and values.
lcolor	the color(s) to be used for the horizontal lines.
xlim	horizontal range for the plot, see plot.window, for example.
main	overall title for the plot, see title.
xlab, ylab	axis annotations as in title.
...	graphical parameters can also be specified as arguments.

Value

This function is invoked for its side effect, which is to produce two variants of dotplots as described in Cleveland (1985).

Dot plots are a reasonable substitute for bar plots.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

Examples

```
dotchart(VADeaths, main = "Death Rates in Virginia - 1940")

op <- par(xaxs = "i")  # 0 -- 100%
dotchart(t(VADeaths), xlim = c(0,100), bg = "skyblue",
         main = "Death Rates in Virginia - 1940", xlab = "rate [ % ]",
         ylab = "Grouping:  Age  x   Urbanity . Gender")
par(op)
```

---

filled.contour *Level (Contour) Plots*

Description

This function produces a contour plot with the areas between the contours filled in solid color (Cleveland calls this a level plot). A key showing how the colors map to z values is shown to the right of the plot.

Usage

```
filled.contour(x = seq(0, 1, length.out = nrow(z)),
               y = seq(0, 1, length.out = ncol(z)),
               z,
               xlim = range(x, finite = TRUE),
               ylim = range(y, finite = TRUE),
               zlim = range(z, finite = TRUE),
               levels = pretty(zlim, nlevels), nlevels = 20,
               color.palette = function(n) hcl.colors(n, "YlOrRd", rev = TRUE),
               col = color.palette(length(levels) - 1),
               plot.title, plot.axes, key.title, key.axes, key.border = NULL,
               asp = NA, xaxs = "i", yaxs = "i", las = 1,
               axes = TRUE, frame.plot = axes, ...)

.filled.contour(x, y, z, levels, col)
```

Arguments

x, y	locations of grid lines at which the values in z are measured. These must be in ascending order. (The rest of this description does not apply to .filled.contour.) By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z.
z	a numeric matrix containing the values to be plotted.. Note that x can be used instead of z for convenience.
xlim	x limits for the plot.
ylim	y limits for the plot.
zlim	z limits for the plot.
levels	a set of levels which are used to partition the range of z. Must be **strictly** increasing (and finite). Areas with z values between consecutive levels are painted with the same color.

nlevels	if levels is not specified, the range of z, values is divided into approximately this many levels.
color.palette	a color palette function to be used to assign colors in the plot.
col	an explicit set of colors to be used in the plot. This argument overrides any palette function specification. There should be one less color than levels
plot.title	statements which add titles to the main plot.
plot.axes	statements which draw axes (and a box) on the main plot. This overrides the default axes.
key.title	statements which add titles for the plot key.
key.axes	statements which draw axes on the plot key. This overrides the default axis.
key.border	color for the border of the key rect()angles.
asp	the y/x aspect ratio, see plot.window.
xaxs	the x axis style. The default is to use internal labeling.
yaxs	the y axis style. The default is to use internal labeling.
las	the style of labeling to be used. The default is to use horizontal labeling.
axes, frame.plot	logicals indicating if axes and a box should be drawn, as in plot.default.
...	additional graphical parameters, currently only passed to title().

Details

The values to be plotted can contain NAs. Rectangles with two or more corner values are NA are omitted entirely: where there is a single NA value the triangle opposite the NA is omitted.

Values to be plotted can be infinite: the effect is similar to that described for NA values.

.filled.contour is a 'bare bones' interface to add just the contour plot to an already-set-up plot region. It is is intended for programmatic use, and the programmer is responsible for checking the conditions on the arguments.

Note

filled.contour uses the layout function and so is restricted to a full page display.

The output produced by filled.contour is actually a combination of two plots; one is the filled contour and one is the legend. Two separate coordinate systems are set up for these two plots, but they are only used internally – once the function has returned these coordinate systems are lost. If you want to annotate the main contour plot, for example to add points, you can specify graphics commands in the plot.axes argument. See the examples.

Author(s)

Ross Ihaka and R Core Team

References

Cleveland, W. S. (1993) *Visualizing Data*. Summit, New Jersey: Hobart.

See Also

contour, image, hcl.colors, gray.colors, palette; contourplot and levelplot from package **lattice**.

Examples

```
require("grDevices") # for colours
filled.contour(volcano, asp = 1) # simple

x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
filled.contour(x, y, volcano,
    color.palette = function(n) hcl.colors(n, "terrain"),
    plot.title = title(main = "The Topography of Maunga Whau",
    xlab = "Meters North", ylab = "Meters West"),
    plot.axes = { axis(1, seq(100, 800, by = 100))
                  axis(2, seq(100, 600, by = 100)) },
    key.title = title(main = "Height\n(meters)"),
    key.axes = axis(4, seq(90, 190, by = 10)))  # maybe also asp = 1
mtext(paste("filled.contour(.) from", R.version.string),
      side = 1, line = 4, adj = 1, cex = .66)

# Annotating a filled contour plot
a <- expand.grid(1:20, 1:20)
b <- matrix(a[,1] + a[,2], 20)
filled.contour(x = 1:20, y = 1:20, z = b,
               plot.axes = { axis(1); axis(2); points(10, 10) })

## Persian Rug Art:
x <- y <- seq(-4*pi, 4*pi, length.out = 27)
r <- sqrt(outer(x^2, y^2, `+`))
## "minimal"
filled.contour(cos(r^2)*exp(-r/(2*pi)), axes = FALSE, key.border=NA)
## rather, the key *should* be labeled (but axes still not):
filled.contour(cos(r^2)*exp(-r/(2*pi)), frame.plot = FALSE,
               plot.axes = {})
```

---

fourfoldplot *Fourfold Plots*

Description

Creates a fourfold display of a 2 by 2 by k contingency table on the current graphics device, allowing for the visual inspection of the association between two dichotomous variables in one or several populations (strata).

Usage

```
fourfoldplot(x, color = c("#99CCFF", "#6699CC"),
             conf.level = 0.95,
             std = c("margins", "ind.max", "all.max"),
             margin = c(1, 2), space = 0.2, main = NULL,
             mfrow = NULL, mfcol = NULL)
```

Arguments

x	a 2 by 2 by k contingency table in array form, or as a 2 by 2 matrix if k is 1.

color	a vector of length 2 specifying the colors to use for the smaller and larger diagonals of each 2 by 2 table.
conf.level	confidence level used for the confidence rings on the odds ratios. Must be a single nonnegative number less than 1; if set to 0, confidence rings are suppressed.
std	a character string specifying how to standardize the table. Must match one of "margins", "ind.max", or "all.max", and can be abbreviated to the initial letter. If set to "margins", each 2 by 2 table is standardized to equate the margins specified by margin while preserving the odds ratio. If "ind.max" or "all.max", the tables are either individually or simultaneously standardized to a maximal cell frequency of 1.
margin	a numeric vector with the margins to equate. Must be one of 1, 2, or c(1, 2) (the default), which corresponds to standardizing the row, column, or both margins in each 2 by 2 table. Only used if std equals "margins".
space	the amount of space (as a fraction of the maximal radius of the quarter circles) used for the row and column labels.
main	character string for the fourfold title.
mfrow	a numeric vector of the form c(nr, nc), indicating that the displays for the 2 by 2 tables should be arranged in an nr by nc layout, filled by rows.
mfcol	a numeric vector of the form c(nr, nc), indicating that the displays for the 2 by 2 tables should be arranged in an nr by nc layout, filled by columns.

Details

The fourfold display is designed for the display of 2 by 2 by k tables.

Following suitable standardization, the cell frequencies f_{ij} of each 2 by 2 table are shown as a quarter circle whose radius is proportional to $\sqrt{f_{ij}}$ so that its area is proportional to the cell frequency. An association (odds ratio different from 1) between the binary row and column variables is indicated by the tendency of diagonally opposite cells in one direction to differ in size from those in the other direction; color is used to show this direction. Confidence rings for the odds ratio allow a visual test of the null of no association; the rings for adjacent quadrants overlap if and only if the observed counts are consistent with the null hypothesis.

Typically, the number k corresponds to the number of levels of a stratifying variable, and it is of interest to see whether the association is homogeneous across strata. The fourfold display visualizes the pattern of association. Note that the confidence rings for the individual odds ratios are not adjusted for multiple testing.

References

Friendly, M. (1994). A fourfold display for 2 by 2 by k tables. Technical Report 217, York University, Psychology Department. http://datavis.ca/papers/4fold/4fold.pdf

See Also

mosaicplot

Examples

```
## Use the Berkeley admission data as in Friendly (1995).
x <- aperm(UCBAdmissions, c(2, 1, 3))
dimnames(x)[[2]] <- c("Yes", "No")
names(dimnames(x)) <- c("Sex", "Admit?", "Department")
```

```
stats::ftable(x)

## Fourfold display of data aggregated over departments, with
## frequencies standardized to equate the margins for admission
## and sex.
## Figure 1 in Friendly (1994).
fourfoldplot(marginSums(x, c(1, 2)))

## Fourfold display of x, with frequencies in each table
## standardized to equate the margins for admission and sex.
## Figure 2 in Friendly (1994).
fourfoldplot(x)

## Fourfold display of x, with frequencies in each table
## standardized to equate the margins for admission. but not
## for sex.
## Figure 3 in Friendly (1994).
fourfoldplot(x, margin = 2)
```

`frame` *Create / Start a New Plot Frame*

Description

This function (`frame` is an alias for `plot.new`) causes the completion of plotting in the current plot (if there is one) and an advance to a new graphics frame. This is used in all high-level plotting functions and also useful for skipping plots when a multi-figure region is in use.

Usage

```
plot.new()
frame()
```

Details

The new page is painted with the background colour (`par("bg")`), which is often transparent. For devices with a *canvas* colour (the on-screen devices `X11`, `windows` and `quartz`), the window is first painted with the canvas colour and then the background colour.

There are two hooks called `"before.plot.new"` and `"plot.new"` (see `setHook`) called immediately before and after advancing the frame. The latter is used in the testing code to annotate the new page. The hook function(s) are called with no argument. (If the value is a character string, `get` is called on it from within the **graphics** namespace.)

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (`frame`.)

See Also

`plot.window`, `plot.default`.

grid *Add Grid to a Plot*

Description

grid adds an nx by ny rectangular grid to an existing plot.

Usage

```
grid(nx = NULL, ny = nx, col = "lightgray", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```

Arguments

nx, ny	number of cells of the grid in x and y direction. When NULL, as per default, the grid aligns with the tick marks on the corresponding *default* axis (i.e., tickmarks as computed by axTicks). When NA, no grid lines are drawn in the corresponding direction.
col	character or (integer) numeric; color of the grid lines.
lty	character or (integer) numeric; line type of the grid lines.
lwd	non-negative numeric giving line width of the grid lines.
equilogs	logical, only used when *log* coordinates and alignment with the axis tick marks are active. Setting equilogs = FALSE in that case gives *non equidistant* tick aligned grid lines.

Note

If more fine tuning is required, use abline(h = ., v = .) directly.

References

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot, abline, lines, points.

Examples

```
plot(1:3)
grid(NA, 5, lwd = 2) # grid only in y-direction

## maybe change the desired number of tick marks:  par(lab = c(mx, my, 7))
op <- par(mfcol = 1:2)
with(iris,
     {
     plot(Sepal.Length, Sepal.Width, col = as.integer(Species),
          xlim = c(4, 8), ylim = c(2, 4.5), panel.first = grid(),
          main = "with(iris,  plot(...., panel.first = grid(), ..) )")
     plot(Sepal.Length, Sepal.Width, col = as.integer(Species),
          panel.first = grid(3, lty = 1, lwd = 2),
          main = "... panel.first = grid(3, lty = 1, lwd = 2), ..")
```

```
     }
    )
par(op)

plot(1:64)
gr <- grid() # now *invisibly* returns the grid "at" locations
str(gr)
stopifnot(length(gr) == 2, identical(gr[[1]], gr[[2]]),
          gr[["atx"]] == 10*(0:6))

## In log-scale plots :
plot(8:270, log="xy") ; grid() # at (1, 10, 100); if preferring "all" grid lines:
plot(8:270, log="xy") ; grid(equilogs = FALSE) -> grll
stopifnot(identical(grll, list(atx = c(1, 2, 5, 10, 20, 50, 100, 200),
                               aty = c(         10, 20, 50, 100, 200))))
```

`hist` *Histograms*

Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE, fuzz = 1e-7,
     density = NULL, angle = 45, col = "lightgray", border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

Arguments

`x`	a vector of values for which the histogram is desired.
`breaks`	one of:

- a vector giving the breakpoints between histogram cells,
- a function to compute the vector of breakpoints,
- a single number giving the number of cells for the histogram,
- a character string naming an algorithm to compute the number of cells (see 'Details'),
- a function to compute the number of cells.

	In the last three cases the number is a suggestion only; as the breakpoints will be set to `pretty` values, the number is limited to `1e6` (with a warning if it was larger). If `breaks` is a function, the `x` vector is supplied to it as the only argument (and the number of breaks is only limited by the amount of available memory).
`freq`	logical; if `TRUE`, the histogram graphic is a representation of frequencies, the `counts` component of the result; if `FALSE`, probability densities, component `density`, are plotted (so that the histogram has a total area of one). Defaults to `TRUE` *if and only if* `breaks` are equidistant (and `probability` is not specified).
`probability`	an *alias* for `!freq`, for S compatibility.
`include.lowest`	logical; if `TRUE`, an `x[i]` equal to the `breaks` value will be included in the first (or last, for `right = FALSE`) bar. This will be ignored (with a warning) unless `breaks` is a vector.
`right`	logical; if `TRUE`, the histogram cells are right-closed (left open) intervals.
`fuzz`	non-negative number, for the case when the data is "pretty" and some observations `x[.]` are close but not exactly on a `break`. For counting fuzzy breaks proportional to `fuzz` are used. The default is occasionally suboptimal.
`density`	the density of shading lines, in lines per inch. The default value of `NULL` means that no shading lines are drawn. Non-positive values of `density` also inhibit the drawing of shading lines.
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise).
`col`	a colour to be used to fill the bars.
`border`	the color of the border around the bars. The default is to use the standard foreground color.
`main`, `xlab`, `ylab`	main title and axis labels: these arguments to `title()` get "smart" defaults here, e.g., the default `ylab` is `"Frequency"` iff `freq` is true.
`xlim`, `ylim`	the range of x and y values with sensible defaults. Note that `xlim` is *not* used to define the histogram (breaks), but only for plotting (when `plot = TRUE`).
`axes`	logical. If `TRUE` (default), axes are draw if the plot is drawn.
`plot`	logical. If `TRUE` (default), a histogram is plotted, otherwise a list of breaks and counts is returned. In the latter case, a warning is used if (typically graphical) arguments are specified that only apply to the `plot = TRUE` case.
`labels`	logical or character string. Additionally draw labels on top of bars, if not `FALSE`; see `plot.histogram`.
`nclass`	numeric (integer). For S(-PLUS) compatibility only, `nclass` is equivalent to `breaks` for a scalar or character argument.
`warn.unused`	logical. If `plot = FALSE` and `warn.unused = TRUE`, a warning will be issued when graphical parameters are passed to `hist.default()`.
`...`	further arguments and graphical parameters passed to `plot.histogram` and thence to `title` and `axis` (if `plot = TRUE`).

Details

The definition of *histogram* differs by source (with country-specific biases). R's default with equispaced breaks (also the default) is to plot the counts in the cells defined by `breaks`. Thus the height of a rectangle is proportional to the number of points falling into the cell, as is the area *provided* the breaks are equally-spaced.

The default with non-equispaced breaks is to give a plot of area one, in which the *area* of the rectangles is the fraction of the data points falling in the cells.

If `right = TRUE` (default), the histogram cells are intervals of the form $(a, b]$, i.e., they include their right-hand endpoint, but not their left one, with the exception of the first cell when `include.lowest` is `TRUE`.

For `right = FALSE`, the intervals are of the form $[a, b)$, and `include.lowest` means '*include highest*'.

A numerical tolerance of 10^{-7} times the median bin size (for more than four bins, otherwise the median is substituted) is applied when counting entries on the edges of bins. This is not included in the reported `breaks` nor in the calculation of `density`.

The default for `breaks` is `"Sturges"`: see `nclass.Sturges`. Other names for which algorithms are supplied are `"Scott"` and `"FD"` / `"Freedman-Diaconis"` (with corresponding functions `nclass.scott` and `nclass.FD`). Case is ignored and partial matching is used. Alternatively, a function can be supplied which will compute the intended number of breaks or the actual breakpoints as a function of `x`.

Value

an object of class `"histogram"` which is a list with components:

`breaks`	the $n + 1$ cell boundaries (= `breaks` if that was a vector). These are the nominal breaks, not with the boundary fuzz.
`counts`	n integers; for each cell, the number of `x[]` inside.
`density`	values $\hat{f}(x_i)$, as estimated density values. If `all(diff(breaks) == 1)`, they are the relative frequencies `counts/n` and in general satisfy $\sum_i \hat{f}(x_i)(b_{i+1} - b_i) = 1$, where b_i = `breaks[i]`.
`mids`	the n cell midpoints.
`xname`	a character string with the actual `x` argument name.
`equidist`	logical, indicating if the distances between `breaks` are all the same.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Venables, W. N. and Ripley. B. D. (2002) *Modern Applied Statistics with S*. Springer.

See Also

`nclass.Sturges`, `stem`, `density`, `truehist` in package **MASS**.

Typical plots with vertical bars are *not* histograms. Consider `barplot` or `plot(*, type = "h")` for such bar plots.

Examples

```
op <- par(mfrow = c(2, 2))
hist(islands)
utils::str(hist(islands, col = "gray", labels = TRUE))

hist(sqrt(islands), breaks = 12, col = "lightblue", border = "pink")
##-- For non-equidistant breaks, counts should NOT be graphed unscaled:
r <- hist(sqrt(islands), breaks = c(4*0:5, 10*3:5, 70, 100, 140),
```

```
          col = "blue1")
text(r$mids, r$density, r$counts, adj = c(.5, -.5), col = "blue3")
sapply(r[2:3], sum)
sum(r$density * diff(r$breaks)) # == 1
lines(r, lty = 3, border = "purple") # -> lines.histogram(*)
par(op)

require(utils) # for str
str(hist(islands, breaks = 12, plot =  FALSE)) #-> 10 (~= 12) breaks
str(hist(islands, breaks = c(12,20,36,80,200,1000,17000), plot = FALSE))

hist(islands, breaks = c(12,20,36,80,200,1000,17000), freq = TRUE,
     main = "WRONG histogram") # and warning

## Extreme outliers; the "FD" rule would take very large number of 'breaks':
XXL <- c(1:9, c(-1,1)*1e300)
hh <- hist(XXL, "FD") # did not work in R <= 3.4.1; now gives warning
## pretty() determines how many counts are used (platform dependently!):
length(hh$breaks) ## typically 1 million -- though 1e6 was "a suggestion only"

## R >= 4.2.0: no "*.5" labels on y-axis:
hist(c(2,3,3,5,5,6,6,6,7))

require(stats)
set.seed(14)
x <- rchisq(100, df = 4)

## Histogram with custom x-axis:
hist(x, xaxt = "n")
axis(1, at = 0:17)


## Comparing data with a model distribution should be done with qqplot()!
qqplot(x, qchisq(ppoints(x), df = 4)); abline(0, 1, col = 2, lty = 2)

## if you really insist on using hist() ... :
hist(x, freq = FALSE, ylim = c(0, 0.2))
curve(dchisq(x, df = 4), col = 2, lty = 2, lwd = 2, add = TRUE)
```

`hist.POSIXt` *Histogram of a Date or Date-Time Object*

Description

Methods for `hist` applied to date (class "`Date`") or date-time (class "`POSIXt`") objects.

Usage

```
## S3 method for class 'POSIXt'
hist(x, breaks, ...,
     xlab = deparse1(substitute(x)),
     plot = TRUE, freq = FALSE,
     start.on.monday = TRUE, format, right = TRUE)
```

```
## S3 method for class 'Date'
hist(x, breaks, ...,
     xlab = deparse1(substitute(x)),
     plot = TRUE, freq = FALSE,
     start.on.monday = TRUE, format, right = TRUE)
```

Arguments

x	an object inheriting from class "POSIXt" or "Date".
breaks	a vector of cut points *or* number giving the number of intervals which x is to be cut into *or* an interval specification, one of "days", "weeks", "months", "quarters" or "years", plus "secs", "mins", "hours" for date-time objects.
...	graphical parameters, or arguments to hist.default such as include.lowest, density and labels.
xlab	a character string giving the label for the x axis, if plotted.
plot	logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned.
freq	logical; if TRUE, the histogram graphic is a representation of frequencies, i.e, the counts component of the result; if FALSE, *relative* frequencies (probabilities) are plotted.
start.on.monday	logical. If breaks = "weeks", should the week start on Mondays or Sundays?
format	for the x-axis labels. See strptime.
right	logical; if TRUE, the histogram cells are right-closed (left open) intervals.

Details

Note that unlike the default method, breaks is a required argument.

Using breaks = "quarters" will create intervals of 3 calendar months, with the intervals beginning on January 1, April 1, July 1 or October 1, based upon min(x) as appropriate.

With the default right = TRUE, breaks will be set on the last day of the previous period when breaks is "months", "quarters" or "years". Use right = FALSE to set them to the first day of the interval shown in each bar.

Value

An object of class "histogram": see hist.

See Also

seq.POSIXt, axis.POSIXct, hist

Examples

```
hist(.leap.seconds, "years", freq = TRUE)
brks <- seq(ISOdate(1970, 1, 1), ISOdate(2030, 1, 1), "5 years")
hist(.leap.seconds, brks)
rug(.leap.seconds, lwd=2)
## show that  'include.lowest' "works"
stopifnot(identical(c(2L, rep(1L,11)),
```

```
   hist(brks, brks, plot=FALSE, include.lowest=TRUE )$counts))
tools::assertError(verbose=TRUE, ##--> 'breaks' do not span range of 'x'
   hist(brks, brks, plot=FALSE, include.lowest=FALSE))
## The default fuzz in hist.default()  "kills" this, with a "wrong" message:
try ( hist(brks[-13] + 1, brks, include.lowest = FALSE) )
## and decreasing 'fuzz' solves the issue:
hb <- hist(brks[-13] + 1, brks, include.lowest = FALSE, fuzz = 1e-10)
stopifnot(hb$counts == 1)

## 100 random dates in a 10-week period
random.dates <- as.Date("2001/1/1") + 70*stats::runif(100)
hist(random.dates, "weeks", format = "%d %b")
```

`identify` *Identify Points in a Scatter Plot*

Description

`identify` reads the position of the graphics pointer when the (first) mouse button is pressed. It then searches the coordinates given in `x` and `y` for the point closest to the pointer. If this point is close enough to the pointer, its index will be returned as part of the value of the call.

Usage

```
identify(x, ...)

## Default S3 method:
identify(x, y = NULL, labels = seq_along(x), pos = FALSE,
         n = length(x), plot = TRUE, atpen = FALSE, offset = 0.5,
         tolerance = 0.25, order = FALSE, ...)
```

Arguments

`x, y`	coordinates of points in a scatter plot. Alternatively, any object which defines coordinates (a plotting structure, time series etc: see `xy.coords`) can be given as `x`, and `y` left missing.
`labels`	an optional character vector giving labels for the points. Will be coerced using `as.character`, and recycled if necessary to the length of `x`. Excess labels will be discarded, with a warning.
`pos`	if `pos` is `TRUE`, a component is added to the return value which indicates where text was plotted relative to each identified point: see Value.
`n`	the maximum number of points to be identified.
`plot`	logical: if `plot` is `TRUE`, the labels are printed near the points and if `FALSE` they are omitted.
`atpen`	logical: if `TRUE` and `plot = TRUE`, the lower-left corners of the labels are plotted at the points clicked rather than relative to the points.
`offset`	the distance (in character widths) which separates the label from identified points. Negative values are allowed. Not used if `atpen = TRUE`.
`tolerance`	the maximal distance (in inches) for the pointer to be 'close enough' to a point.
`order`	if `order` is `TRUE`, a component is added to the return value which indicates the order in which points were identified: see Value.
`...`	further arguments passed to `par` such as `cex`, `col` and `font`.

Details

identify is a generic function, and only the default method is described here.

identify is only supported on screen devices such as X11, windows and quartz. On other devices the call will do nothing.

Clicking near (as defined by tolerance) a point adds it to the list of identified points. Points can be identified only once, and if the point has already been identified or the click is not near any of the points a message is printed immediately on the R console.

If plot is TRUE, the point is labelled with the corresponding element of labels. If atpen is false (the default) the labels are placed below, to the left, above or to the right of the identified point, depending on where the pointer was relative to the point. If atpen is true, the labels are placed with the bottom left of the string's box at the pointer.

For the usual X11 device the identification process is terminated by pressing any mouse button other than the first. For the quartz device the process is terminated by pressing either the pop-up menu equivalent (usually second mouse button or Ctrl-click) or the ESC key.

On most devices which support identify, successful selection of a point is indicated by a bell sound unless options(locatorBell = FALSE) has been set.

If the window is resized or hidden and then exposed before the identification process has terminated, any labels drawn by identify will disappear. These will reappear once the identification process has terminated and the window is resized or hidden and exposed again. This is because the labels drawn by identify are not recorded in the device's display list until the identification process has terminated.

If you interrupt the identify call this leaves the graphics device in an undefined state, with points labelled but labels not recorded in the display list. Copying a device in that state will give unpredictable results.

Value

If both pos and order are FALSE, an integer vector containing the indices of the identified points.

If either of pos or order is TRUE, a list containing a component ind, indicating which points were identified and (if pos is TRUE) a component pos, indicating where the labels were placed relative to the identified points (1=below, 2=left, 3=above, 4=right and 0=no offset, used if atpen = TRUE) and (if order is TRUE) a component order, indicating the order in which points were identified.

Technicalities

The algorithm used for placing labels is the same as used by text if pos is specified there, the difference being that the position of the pointer relative the identified point determines pos in identify.

For labels placed to the left of a point, the right-hand edge of the string's box is placed offset units to the left of the point, and analogously for points to the right. The baseline of the text is placed below the point so as to approximately centre string vertically. For labels placed above or below a point, the string is centered horizontally on the point. For labels placed above, the baseline of the text is placed offset units above the point, and for those placed below, the baseline is placed so that the top of the string's box is approximately offset units below the point. If you want more precise placement (e.g., centering) use plot = FALSE and plot via text or points: see the examples.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

locator, text.

dev.capabilities to see if it is supported.

Examples

```
## A function to use identify to select points, and overplot the
## points with another symbol as they are selected
identifyPch <- function(x, y = NULL, n = length(x), plot = FALSE, pch = 19, ...)
{
    xy <- xy.coords(x, y); x <- xy$x; y <- xy$y
    sel <- rep(FALSE, length(x))
    while(sum(sel) < n) {
        ans <- identify(x[!sel], y[!sel], labels = which(!sel), n = 1, plot = plot, ...)
        if(!length(ans)) break
        ans <- which(!sel)[ans]
        points(x[ans], y[ans], pch = pch)
        sel[ans] <- TRUE
    }
    ## return indices of selected points
    which(sel)
}

if(dev.interactive()) { ## use it
  x <- rnorm(50); y <- rnorm(50)
  plot(x,y); identifyPch(x,y) # how fast to get all?
}
```

---

image *Display a Color Image*

Description

Creates a grid of colored or gray-scale rectangles with colors corresponding to the values in z. This can be used to display three-dimensional or spatial data aka *images*. This is a generic function.

NOTE: the grid is drawn as a set of rectangles by default; see the useRaster argument to draw the grid as a raster image.

The function hcl.colors provides a broad range of sequential color palettes that are suitable for displaying ordered data, with n giving the number of colors desired.

Usage

```
image(x, ...)

## Default S3 method:
image(x, y, z, zlim, xlim, ylim,
      col = hcl.colors(12, "YlOrRd", rev = TRUE),
      add = FALSE, xaxs = "i", yaxs = "i", xlab, ylab,
      breaks, oldstyle = FALSE, useRaster, ...)
```

Arguments

x, y	locations of grid lines at which the values in z are measured. These must be finite, non-missing and in (strictly) ascending order. By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z.
z	a numeric or logical matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience.
zlim	the minimum and maximum z values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The *midpoints* of the intervals cover the range, so that values just outside the range will be plotted.
xlim, ylim	ranges for the plotted x and y values, defaulting to the ranges of x and y.
col	a list of colors such as that generated by hcl.colors, gray.colors or similar functions.
add	logical; if TRUE, add to current plot (and disregard the following four arguments). This is rarely useful because image 'paints' over existing graphics.
xaxs, yaxs	style of x and y axis. The default "i" is appropriate for images. See par.
xlab, ylab	each a character string giving the labels for the x and y axis. Default to the 'call names' of x or y, or to "" if these were unspecified.
breaks	a set of finite numeric breakpoints for the colours: must have one more breakpoint than colour and be in increasing order. Unsorted vectors will be sorted, with a warning.
oldstyle	logical. If true the midpoints of the colour intervals are equally spaced, and zlim[1] and zlim[2] were taken to be midpoints. The default is to have colour intervals of equal lengths between the limits.
useRaster	logical; if TRUE a bitmap raster is used to plot the image instead of polygons. The grid must be regular in that case, otherwise an error is raised. For the behaviour when this is not specified, see 'Details'.
...	graphical parameters for plot may also be passed as arguments to this function, as can the plot aspect ratio asp and axes (see plot.window).

Details

The length of x should be equal to the nrow(z)+1 or nrow(z). In the first case x specifies the boundaries between the cells: in the second case x specifies the midpoints of the cells. Similar reasoning applies to y. It probably only makes sense to specify the midpoints of an equally-spaced grid. If you specify just one row or column and a length-one x or y, the whole user area in the corresponding direction is filled. For logarithmic x or y axes the boundaries between cells must be specified.

Rectangles corresponding to missing values are not plotted (and so are transparent and (unless add = TRUE) the default background painted in par("bg") will show through and if that is transparent, the canvas colour will be seen).

If breaks is specified then zlim is unused and the algorithm used follows cut, so intervals are closed on the right and open on the left except for the lowest interval which is closed at both ends.

The axes (where plotted) make use of the classes of xlim and ylim (and hence by default the classes of x and y): this will mean that for example dates are labelled as such.

Notice that image interprets the z matrix as a table of f(x[i], y[j]) values, so that the x axis corresponds to row number and the y axis to column number, with column 1 at the bottom, i.e. a 90 degree counter-clockwise rotation of the conventional printed layout of a matrix.

Images for large z on a regular grid are rendered more efficiently with useRaster = TRUE and can prevent rare anti-aliasing artifacts, but may not be supported by all graphics devices. Some devices (such as postscript and X11(type = "Xlib")) which do not support semi-transparent colours may emit missing values as white rather than transparent, and there may be limitations on the size of a raster image. (Problems with the rendering of raster images have been reported by users of windows() devices under Remote Desktop, at least under its default settings.)

The graphics files in PDF and PostScript can be much smaller under this option.

If useRaster is not specified, raster images are used when the getOption("preferRaster") is true, the grid is regular and either dev.capabilities("rasterImage")$rasterImage is "yes" or it is "non-missing" and there are no missing values.

Note

Originally based on a function by Thomas Lumley.

See Also

filled.contour or heatmap which can look nicer (but are less modular), contour; The **lattice** equivalent of image is levelplot.

hcl.colors, gray.colors, hcl, hsv, par.

dev.capabilities to see if useRaster = TRUE is supported on the current device.

Examples

```
require("grDevices") # for colours
x <- y <- seq(-4*pi, 4*pi, length.out = 27)
r <- sqrt(outer(x^2, y^2, `+`))
image(z = z <- cos(r^2)*exp(-r/6), col = gray.colors(33))
image(z, axes = FALSE, main = "Math can be beautiful ...",
      xlab = expression(cos(r^2) * e^{-r/6}))
contour(z, add = TRUE, drawlabels = FALSE)

# Visualize as matrix.  Need to transpose matrix and then flip it horizontally:
tf <- function(m) t(m)[, nrow(m):1]
imageM <- function(m, grid = max(dim(m)) <= 25, asp = (nrow(m)-1)/(ncol(m)-1), ...) {
    image(tf(m), asp=asp, axes = FALSE, ...)
    mAxis <- function(side, at, ...) # using 'j'
        axis(side, at=at, labels=as.character(j+1L), col="gray", col.axis=1, ...)
    n <- ncol(m); n1 <- n-1L; j <- 0L:n1; mAxis(1, at= j/n1)
    if(grid) abline(v = (0:n - .5)/n1, col="gray77", lty="dotted")
    n <- nrow(m); n1 <- n-1L; j <- 0L:n1; mAxis(2, at=1-j/n1, las=1)
    if(grid) abline(h = (0:n - .5)/n1, col="gray77", lty="dotted")
}
(m <- outer(1:5, 1:14))
imageM(m, main = "image(<5 x 14 matrix>)   with rows and columns")
imageM(volcano)

# A prettier display of the volcano
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))
image(x, y, volcano, col = hcl.colors(100, "terrain"), axes = FALSE)
```

```
contour(x, y, volcano, levels = seq(90, 200, by = 5),
        add = TRUE, col = "brown")
axis(1, at = seq(100, 800, by = 100))
axis(2, at = seq(100, 600, by = 100))
box()
title(main = "Maunga Whau Volcano", font.main = 4)
```

---

layout *Specifying Complex Plot Arrangements*

Description

`layout` divides the device up into as many rows and columns as there are in matrix `mat`, with the column-widths and the row-heights specified in the respective arguments.

Usage

```
layout(mat, widths = rep.int(1, ncol(mat)),
       heights = rep.int(1, nrow(mat)), respect = FALSE)

layout.show(n = 1)
lcm(x)
```

Arguments

`mat`	a matrix object specifying the location of the next N figures on the output device. Each value in the matrix must be `0` or a positive integer. If N is the largest positive integer in the matrix, then the integers $\{1, \ldots, N-1\}$ must also appear at least once in the matrix.
`widths`	a vector of values for the widths of columns on the device. Relative widths are specified with numeric values. Absolute widths (in centimetres) are specified with the `lcm()` function (see examples).
`heights`	a vector of values for the heights of rows on the device. Relative and absolute heights can be specified, see `widths` above.
`respect`	either a logical value or a matrix object. If the latter, then it must have the same dimensions as `mat` and each value in the matrix must be either `0` or `1`.
`n`	number of figures to plot.
`x`	a dimension to be interpreted as a number of centimetres.

Details

Figure i is allocated a region composed from a subset of these rows and columns, based on the rows and columns in which i occurs in `mat`.

The `respect` argument controls whether a unit column-width is the same physical measurement on the device as a unit row-height.

There is a limit (currently 200) for the numbers of rows and columns in the layout, and also for the total number of cells (10007).

`layout.show(n)` plots (part of) the current layout, namely the outlines of the next `n` figures.

`lcm` is a trivial function, to be used as *the* interface for specifying absolute dimensions for the `widths` and `heights` arguments of `layout()`.

Value

`layout` returns the number of figures, N, see above.

Warnings

These functions are totally incompatible with the other mechanisms for arranging plots on a device: `par(mfrow)`, `par(mfcol)` and `split.screen`.

Author(s)

Paul R. Murrell

References

Murrell, P. R. (1999). Layouts: A mechanism for arranging plots on a page. *Journal of Computational and Graphical Statistics*, **8**, 121–134. doi:10.2307/1390924.

Chapter 5 of Paul Murrell's Ph.D. thesis.

Murrell, P. (2005). *R Graphics*. Chapman & Hall/CRC Press.

See Also

`par` with arguments `mfrow`, `mfcol`, or `mfg`.

Examples

```
def.par <- par(no.readonly = TRUE) # save default, for resetting...

## divide the device into two rows and two columns
## allocate figure 1 all of row 1
## allocate figure 2 the intersection of column 2 and row 2
layout(matrix(c(1,1,0,2), 2, 2, byrow = TRUE))
## show the regions that have been allocated to each plot
layout.show(2)

## divide device into two rows and two columns
## allocate figure 1 and figure 2 as above
## respect relations between widths and heights
nf <- layout(matrix(c(1,1,0,2), 2, 2, byrow = TRUE), respect = TRUE)
layout.show(nf)

## create single figure which is 5cm square
nf <- layout(matrix(1), widths = lcm(5), heights = lcm(5))
layout.show(nf)


##-- Create a scatterplot with marginal histograms -----

x <- pmin(3, pmax(-3, stats::rnorm(50)))
y <- pmin(3, pmax(-3, stats::rnorm(50)))
xhist <- hist(x, breaks = seq(-3,3,0.5), plot = FALSE)
yhist <- hist(y, breaks = seq(-3,3,0.5), plot = FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(-3, 3)
yrange <- c(-3, 3)
nf <- layout(matrix(c(2,0,1,3),2,2,byrow = TRUE), c(3,1), c(1,3), TRUE)
```

```
layout.show(nf)

par(mar = c(3,3,1,1))
plot(x, y, xlim = xrange, ylim = yrange, xlab = "", ylab = "")
par(mar = c(0,3,1,1))
barplot(xhist$counts, axes = FALSE, ylim = c(0, top), space = 0)
par(mar = c(3,0,1,1))
barplot(yhist$counts, axes = FALSE, xlim = c(0, top), space = 0, horiz = TRUE)

par(def.par)  #- reset to default
```

`legend` *Add Legends to Plots*

Description

This function can be used to add legends to plots. Note that a call to the function `locator(1)` can be used in place of the `x` and `y` arguments.

Usage

```
legend(x, y = NULL, legend, fill = NULL, col = par("col"),
       border = "black", lty, lwd, pch,
       angle = 45, density = NULL, bty = "o", bg = par("bg"),
       box.lwd = par("lwd"), box.lty = par("lty"), box.col = par("fg"),
       pt.bg = NA, cex = 1, pt.cex = cex, pt.lwd = lwd,
       xjust = 0, yjust = 1, x.intersp = 1, y.intersp = 1,
       adj = c(0, 0.5), text.width = NULL, text.col = par("col"),
       text.font = NULL, merge = do.lines && has.pch, trace = FALSE,
       plot = TRUE, ncol = 1, horiz = FALSE, title = NULL,
       inset = 0, xpd, title.col = text.col[1], title.adj = 0.5,
       title.cex = cex[1], title.font = text.font[1],
       seg.len = 2)
```

Arguments

`x`, `y`	the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by `xy.coords`: See 'Details'.
`legend`	a character or expression vector of length ≥ 1 to appear in the legend. Other objects will be coerced by `as.graphicsAnnot`.
`fill`	if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.
`col`	the color of points or lines appearing in the legend.
`border`	the border color for the boxes (used only if `fill` is specified).
`lty`, `lwd`	the line types and widths for lines appearing in the legend. One of these two *must* be specified for line drawing.
`pch`	the plotting symbols appearing in the legend, as numeric vector or a vector of 1-character strings (see `points`). Unlike `points`, this can all be specified as a single multi-character string. *Must* be specified for symbol drawing.
`angle`	angle of shading lines.

`density` the density of shading lines, if numeric and positive. If `NULL` or negative or `NA` color filling is assumed.

`bty` the type of box to be drawn around the legend. The allowed values are `"o"` (the default) and `"n"`.

`bg` the background color for the legend box. (Note that this is only used if `bty != "n"`.)

`box.lty, box.lwd, box.col`
the line type, width and color for the legend box (if `bty = "o"`).

`pt.bg` the background color for the `points`, corresponding to its argument `bg`.

`cex` character expansion factor **relative** to current `par("cex")`. Used for text, and provides the default for `pt.cex`.

`pt.cex` expansion factor(s) for the points.

`pt.lwd` line width for the points, defaults to the one for lines, or if that is not set, to `par("lwd")`.

`xjust` how the legend is to be justified relative to the legend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.

`yjust` the same as `xjust` for the legend y location.

`x.intersp` character interspacing factor for horizontal (x) spacing between symbol and legend text.

`y.intersp` vertical (y) distances (in lines of text shared above/below each legend entry). A vector with one element for each row of the legend can be used.

`adj` numeric of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when `labels` are plotmath expressions.

`text.width` the width of the legend text in x (`"user"`) coordinates. (Should be positive even for a reversed x axis.) Can be a single positive numeric value (same width for each column of the legend), a vector (one element for each column of the legend), `NULL` (default) for computing a proper maximum value of `strwidth(legend)`), or `NA` for computing a proper column wise maximum value of `strwidth(legend)`).

`text.col` the color used for the legend text.

`text.font` the font used for the legend text, see `text`.

`merge` logical; if `TRUE`, merge points and lines but not filled boxes. Defaults to `TRUE` if there are points and lines.

`trace` logical; if `TRUE`, shows how `legend` does all its magical computations.

`plot` logical. If `FALSE`, nothing is plotted but the sizes are returned.

`ncol` the number of columns in which to set the legend items (default is 1, a vertical legend).

`horiz` logical; if `TRUE`, set the legend horizontally rather than vertically (specifying `horiz` overrides the `ncol` specification).

`title` a character string or length-one expression giving a title to be placed at the top of the legend. Other objects will be coerced by `as.graphicsAnnot`.

`inset` inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.

`xpd` if supplied, a value of the graphical parameter `xpd` to be used while the legend is being drawn.

title.col	color for title, defaults to text.col[1].
title.adj	horizontal adjustment for title: see the help for par("adj").
title.cex	expansion factor(s) for the title, defaults to cex[1].
title.font	the font used for the legend title, defaults to text.font[1], see text.
seg.len	the length of lines drawn to illustrate lty and/or lwd (in units of character widths).

Details

Arguments x, y, legend are interpreted in a non-standard way to allow the coordinates to be specified *via* one or two arguments. If legend is missing and y is not numeric, it is assumed that the second argument is intended to be legend and that the first argument specifies the coordinates.

The coordinates can be specified in any way which is accepted by xy.coords. If this gives the coordinates of one point, it is used as the top-left coordinate of the rectangle containing the legend. If it gives the coordinates of two points, these specify opposite corners of the rectangle (either pair of corners, in any order).

The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location. Partial argument matching is used. The optional inset argument specifies how far the legend is inset from the plot margins. If a single value is given, it is used for both margins; if two values are given, the first is used for x-distance, the second for y-distance.

Attribute arguments such as col, pch, lty, etc, are recycled if necessary: merge is not. Set entries of lty to 0 or set entries of lwd to NA to suppress lines in corresponding legend entries; set pch values to NA to suppress points.

Points are drawn *after* lines in order that they can cover the line with their background color pt.bg, if applicable.

See the examples for how to right-justify labels.

Since they are not used for Unicode code points, values -31:-1 are silently omitted, as are NA and "" values.

Value

A list with list components

rect	a list with components
	w, h positive numbers giving **w**idth and **h**eight of the legend's box.
	left, top x and y coordinates of upper left corner of the box.
text	a list with components
	x, y numeric vectors of length length(legend), giving the x and y coordinates of the legend's text(s).

returned invisibly.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot, barplot which uses legend(), and text for more examples of math expressions.

Examples

```
## Run the example in '?matplot' or the following:
leg.txt <- c("Setosa     Petals", "Setosa     Sepals",
             "Versicolor Petals", "Versicolor Sepals")
y.leg <- c(4.5, 3, 2.1, 1.4, .7)
cexv  <- c(1.2, 1, 4/5, 2/3, 1/2)
matplot(c(1, 8), c(0, 4.5), type = "n", xlab = "Length", ylab = "Width",
        main = "Petal and Sepal Dimensions in Iris Blossoms")
for (i in seq(cexv)) {
  text  (1, y.leg[i] - 0.1, paste("cex=", formatC(cexv[i])), cex = 0.8, adj = 0)
  legend(3, y.leg[i], leg.txt, pch = "sSvV", col = c(1, 3), cex = cexv[i])
}
## cex *vector* [in R <= 3.5.1 has 'if(xc < 0)' w/ length(xc) == 2]
legend("right", leg.txt, pch = "sSvV", col = c(1, 3),
       cex = 1+(-1:2)/8, trace = TRUE)# trace: show computed lengths & coords

## 'merge = TRUE' for merging lines & points:
x <- seq(-pi, pi, length.out = 65)
for(reverse in c(FALSE, TRUE)) {  ## normal *and* reverse axes:
  F <- if(reverse) rev else identity
  plot(x, sin(x), type = "l", col = 3, lty = 2,
       xlim = F(range(x)), ylim = F(c(-1.2, 1.8)))
  points(x, cos(x), pch = 3, col = 4)
  lines(x, tan(x), type = "b", lty = 1, pch = 4, col = 6)
  title("legend('top', lty = c(2, -1, 1), pch = c(NA, 3, 4), merge = TRUE)",
        cex.main = 1.1)
  legend("top", c("sin", "cos", "tan"), col = c(3, 4, 6),
       text.col = "green4", lty = c(2, -1, 1), pch = c(NA, 3, 4),
       merge = TRUE, bg = "gray90", trace=TRUE)

} # for(..)

## right-justifying a set of labels: thanks to Uwe Ligges
x <- 1:5; y1 <- 1/x; y2 <- 2/x
plot(rep(x, 2), c(y1, y2), type = "n", xlab = "x", ylab = "y")
lines(x, y1); lines(x, y2, lty = 2)
temp <- legend("topright", legend = c(" ", " "),
               text.width = strwidth("1,000,000"),
               lty = 1:2, xjust = 1, yjust = 1, inset = 1/10,
               title = "Line Types", title.cex = 0.5, trace=TRUE)
text(temp$rect$left + temp$rect$w, temp$text$y,
     c("1,000", "1,000,000"), pos = 2)


##--- log scaled Examples ------------------------------
leg.txt <- c("a one", "a two")

par(mfrow = c(2, 2))
for(ll in c("","x","y","xy")) {
  plot(2:10, log = ll, main = paste0("log = '", ll, "'"))
  abline(1, 1)
  lines(2:3, 3:4, col = 2)
```

```
  points(2, 2, col = 3)
  rect(2, 3, 3, 2, col = 4)
  text(c(3,3), 2:3, c("rect(2,3,3,2, col=4)",
                      "text(c(3,3),2:3,\"c(rect(...)\")"), adj = c(0, 0.3))
  legend(list(x = 2,y = 8), legend = leg.txt, col = 2:3, pch = 1:2,
         lty = 1)  #, trace = TRUE)
} #      ^^^^^^^ to force lines -> automatic merge=TRUE
par(mfrow = c(1,1))

##-- Math expressions:  ------------------------------
x <- seq(-pi, pi, length.out = 65)
plot(x, sin(x), type = "l", col = 2, xlab = expression(phi),
     ylab = expression(f(phi)))
abline(h = -1:1, v = pi/2*(-6:6), col = "gray90")
lines(x, cos(x), col = 3, lty = 2)
ex.cs1 <- expression(plain(sin) * phi,  paste("cos", phi))  # 2 ways
utils::str(legend(-3, .9, ex.cs1, lty = 1:2, plot = FALSE,
           adj = c(0, 0.6)))  # adj y !
legend(-3, 0.9, ex.cs1, lty = 1:2, col = 2:3,  adj = c(0, 0.6))

require(stats)
x <- rexp(100, rate = .5)
hist(x, main = "Mean and Median of a Skewed Distribution")
abline(v = mean(x),   col = 2, lty = 2, lwd = 2)
abline(v = median(x), col = 3, lty = 3, lwd = 2)
ex12 <- expression(bar(x) == sum(over(x[i], n), i == 1, n),
                   hat(x) == median(x[i], i == 1, n))
utils::str(legend(4.1, 30, ex12, col = 2:3, lty = 2:3, lwd = 2))

## 'Filled' boxes -- see also example(barplot) which may call legend(*, fill=)
barplot(VADeaths)
legend("topright", rownames(VADeaths), fill = gray.colors(nrow(VADeaths)))

## Using 'ncol'
x <- 0:64/64
for(R in c(identity, rev)) { # normal *and* reverse x-axis works fine:
  xl <- R(range(x)); x1 <- xl[1]
matplot(x, outer(x, 1:7, function(x, k) sin(k * pi * x)), xlim=xl,
        type = "o", col = 1:7, ylim = c(-1, 1.5), pch = "*")
op <- par(bg = "antiquewhite1")
legend(x1, 1.5, paste("sin(", 1:7, "pi * x)"), col = 1:7, lty = 1:7,
       pch = "*", ncol = 4, cex = 0.8)
legend("bottomright", paste("sin(", 1:7, "pi * x)"), col = 1:7, lty = 1:7,
       pch = "*", cex = 0.8)
legend(x1, -.1, paste("sin(", 1:4, "pi * x)"), col = 1:4, lty = 1:4,
       ncol = 2, cex = 0.8)
legend(x1, -.4, paste("sin(", 5:7, "pi * x)"), col = 4:6,  pch = 24,
       ncol = 2, cex = 1.5, lwd = 2, pt.bg = "pink", pt.cex = 1:3)
par(op)

} # for(..)

## point covering line :
y <- sin(3*pi*x)
plot(x, y, type = "l", col = "blue",
    main = "points with bg & legend(*, pt.bg)")
points(x, y, pch = 21, bg = "white")
```

```
legend(.4,1, "sin(c x)", pch = 21, pt.bg = "white", lty = 1, col = "blue")

## legends with titles at different locations
plot(x, y, type = "n")
legend("bottomright", "(x,y)", pch=1, title= "bottomright")
legend("bottom",      "(x,y)", pch=1, title= "bottom")
legend("bottomleft",  "(x,y)", pch=1, title= "bottomleft")
legend("left",        "(x,y)", pch=1, title= "left")
legend("topleft",     "(x,y)", pch=1, title= "topleft, inset = .05", inset = .05)
legend("top",         "(x,y)", pch=1, title= "top")
legend("topright",    "(x,y)", pch=1, title= "topright, inset = .02",inset = .02)
legend("right",       "(x,y)", pch=1, title= "right")
legend("center",      "(x,y)", pch=1, title= "center")

# using text.font (and text.col):
op <- par(mfrow = c(2, 2), mar = rep(2.1, 4))
c6 <- terrain.colors(10)[1:6]
for(i in 1:4) {
   plot(1, type = "n", axes = FALSE, ann = FALSE); title(paste("text.font =",i))
   legend("top", legend = LETTERS[1:6], col = c6,
          ncol = 2, cex = 2, lwd = 3, text.font = i, text.col = c6)
}
par(op)

# using text.width for several columns
plot(1, type="n")
legend("topleft", c("This legend", "has", "equally sized", "columns."),
       pch = 1:4, ncol = 4)
legend("bottomleft", c("This legend", "has", "optimally sized", "columns."),
       pch = 1:4, ncol = 4, text.width = NA)
legend("right", letters[1:4], pch = 1:4, ncol = 4,
       text.width = 1:4 / 50)
```

`lines` *Add Connected Line Segments to a Plot*

Description

A generic function taking coordinates given in various ways and joining the corresponding points with line segments.

Usage

```
lines(x, ...)

## Default S3 method:
lines(x, y = NULL, type = "l", ...)
```

Arguments

x, y	coordinate vectors of points to join.
type	character indicating the type of plotting; actually any of the types as in `plot.default`.

`...` Further graphical parameters (see `par`) may also be supplied as arguments, particularly, line type, `lty`, line width, `lwd`, color, `col` and for `type = "b"`, `pch`. Also the line characteristics `lend`, `ljoin` and `lmitre`.

Details

The coordinates can be passed in a plotting structure (a list with `x` and `y` components), a two-column matrix, a time series, ... See `xy.coords`. If supplied separately, they must be of the same length.

The coordinates can contain `NA` values. If a point contains `NA` in either its `x` or `y` value, it is omitted from the plot, and lines are not drawn to or from such points. Thus missing values can be used to achieve breaks in lines.

For `type = "h"`, `col` can be a vector and will be recycled as needed.

`lwd` can be a vector: its first element will apply to lines but the whole vector to symbols (recycled as necessary).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`lines.formula` for the formula method; `points`, particularly for `type %in% c("p","b","o")`, `plot`, and the workhorse function `plot.xy`.

`abline` for drawing (single) straight lines.

`par` for line type (`lty`) specification and how to specify colors.

Examples

```
# draw a smooth line through a scatter plot
plot(cars, main = "Stopping Distance versus Speed")
lines(stats::lowess(cars))
```

`locator` *Graphical Input*

Description

Reads the position of the graphics cursor when the (first) mouse button is pressed.

Usage

```
locator(n = 512, type = "n", ...)
```

Arguments

`n`	the maximum number of points to locate. Valid values start at 1.
`type`	One of `"n"`, `"p"`, `"l"` or `"o"`. If `"p"` or `"o"` the points are plotted; if `"l"` or `"o"` they are joined by lines.
`...`	additional graphics parameters used if `type != "n"` for plotting the locations.

Details

`locator` is only supported on screen devices such as `X11`, `windows` and `quartz`. On other devices the call will do nothing.

Unless the process is terminated prematurely by the user (see below) at most `n` positions are determined.

For the usual `X11` device the identification process is terminated by pressing any mouse button other than the first. For the `quartz` device the process is terminated by pressing the `ESC` key.

The current graphics parameters apply just as if `plot.default` has been called with the same value of `type`. The plotting of the points and lines is subject to clipping, but locations outside the current clipping rectangle will be returned.

On most devices which support `locator`, successful selection of a point is indicated by a bell sound unless `options(locatorBell = FALSE)` has been set.

If the window is resized or hidden and then exposed before the input process has terminated, any lines or points drawn by `locator` will disappear. These will reappear once the input process has terminated and the window is resized or hidden and exposed again. This is because the points and lines drawn by `locator` are not recorded in the device's display list until the input process has terminated.

Value

A list containing `x` and `y` components which are the coordinates of the identified points in the user coordinate system, i.e., the one specified by `par("usr")`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`identify`. `grid.locator` is the corresponding **grid** package function.

`dev.capabilities` to see if it is supported.

`matplot` *Plot Columns of Matrices*

Description

Plot the columns of one matrix against the columns of another (which often is just a vector treated as 1-column matrix).

Usage

```
matplot(x, y, type = "p", lty = 1:5, lwd = 1, lend = par("lend"),
        pch = NULL,
        col = 1:6, cex = NULL, bg = NA,
        xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
        log = "", ..., add = FALSE, verbose = getOption("verbose"))
```

```
matpoints(x, y, type = "p", lty = 1:5, lwd = 1, pch = NULL,
          col = 1:6, ...)

matlines (x, y, type = "l", lty = 1:5, lwd = 1, pch = NULL,
          col = 1:6, ...)
```

Arguments

x, y	vectors or matrices of data for plotting. The number of rows should match. If one of them are missing, the other is taken as y and an x vector of 1:n is used. Missing values (NAs) are allowed. Typically, class(.)es of x and y such as "Date" are preserved.
type	character string (length 1 vector) or vector of 1-character strings indicating the type of plot for each column of y, see plot for all possible types. The first character of type defines the first plot, the second character the second, etc. Characters in type are cycled through; e.g., "pl" alternately plots points and lines.
lty, lwd, lend	vector of line types, widths, and end styles. The first element is for the first column, the second element for the second column, etc., even if lines are not plotted for all columns. Line types will be used cyclically until all plots are drawn.
pch	character string or vector of 1-characters or integers for plotting characters, see points for details. The first character is the plotting-character for the first plot, the second for the second, etc. The default is the digits (1 through 9, 0) then the lowercase and uppercase letters.
col	vector of colors. Colors are used cyclically.
cex	vector of character expansion sizes, used cyclically. This works as a multiple of par("cex"). NULL is equivalent to 1.0.
bg	vector of background (fill) colors for the open plot symbols given by pch = 21:25 as in points. The default NA corresponds to the one of the underlying function plot.xy.
xlab, ylab	titles for x and y axes, as in plot.
xlim, ylim	ranges of x and y axes, as in plot.
log, ...	Graphical parameters (see par) and any further arguments of plot, typically plot.default, may also be supplied as arguments to this function; even panel.first etc now work. Hence, the high-level graphics control arguments described under par and the arguments to title may be supplied to this function.
add	logical. If TRUE, plots are added to current one, using points and lines.
verbose	logical. If TRUE, write one line of what is done.

Details

matplot(x,y, ..) is basically a wrapper for

1. calling (the generic function) plot(x[,1], y[,1], ..) for the first columns (only if add = TRUE).
2. calling (the generic) lines(x[,j], y[,j], ..) for subsequent columns.

Care is taken to keep the `class(.)` of `x` and `y`, such that the corresponding `plot()` and `lines()` *methods* will be called.

Points involving missing values are not plotted.

The first column of `x` is plotted against the first column of `y`, the second column of `x` against the second column of `y`, etc. If one matrix has fewer columns, plotting will cycle back through the columns again. (In particular, either `x` or `y` may be a vector, against which all columns of the other argument will be plotted.)

The first element of `col, cex, lty, lwd` is used to plot the axes as well as the first line.

Because plotting symbols are drawn with lines and because these functions may be changing the line style, you should probably specify `lty = 1` when using plotting symbols.

Side Effects

Function `matplot` generates a new plot; `matpoints` and `matlines` add to the current one.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`plot`, `points`, `lines`, `matrix`, `par`.

Examples

```
require(grDevices)
matplot((-4:5)^2, main = "Quadratic") # almost identical to plot(*)
sines <- outer(1:20, 1:4, function(x, y) sin(x / 20 * pi * y))
matplot(sines, pch = 1:4, type = "o", col = rainbow(ncol(sines)))
matplot(sines, type = "b", pch = 21:23, col = 2:5, bg = 2:5,
        main = "matplot(...., pch = 21:23, bg = 2:5)")

x <- 0:50/50
matplot(x, outer(x, 1:8, function(x, k) sin(k*pi * x)),
        ylim = c(-2,2), type = "plobcsSh",
        main= "matplot(,type = \"plobcsSh\" )")
## pch & type =  vector of 1-chars :
matplot(x, outer(x, 1:4, function(x, k) sin(k*pi * x)),
        pch = letters[1:4], type = c("b","p","o"))

lends <- c("round","butt","square")
matplot(matrix(1:12, 4), type="c", lty=1, lwd=10, lend=lends)
text(cbind(2.5, 2*c(1,3,5)-.4), lends, col= 1:3, cex = 1.5)

table(iris$Species) # is data.frame with 'Species' factor
iS <- iris$Species == "setosa"
iV <- iris$Species == "versicolor"
op <- par(bg = "bisque")
matplot(c(1, 8), c(0, 4.5), type =  "n", xlab = "Length", ylab = "Width",
        main = "Petal and Sepal Dimensions in Iris Blossoms")
matpoints(iris[iS,c(1,3)], iris[iS,c(2,4)], pch = "sS", col = c(2,4))
matpoints(iris[iV,c(1,3)], iris[iV,c(2,4)], pch = "vV", col = c(2,4))
legend(1, 4, c("    Setosa Petals", "    Setosa Sepals",
```

```
               "Versicolor Petals", "Versicolor Sepals"),
       pch = "sSvV", col = rep(c(2,4), 2))

nam.var <- colnames(iris)[-5]
nam.spec <- as.character(iris[1+50*0:2, "Species"])
iris.S <- array(NA, dim = c(50,4,3),
                dimnames = list(NULL, nam.var, nam.spec))
for(i in 1:3) iris.S[,,i] <- data.matrix(iris[1:50+50*(i-1), -5])

matplot(iris.S[, "Petal.Length",], iris.S[, "Petal.Width",], pch = "SCV",
        col = rainbow(3, start = 0.8, end = 0.1),
        sub = paste(c("S", "C", "V"), dimnames(iris.S)[[3]],
                    sep = "=", collapse= ",  "),
        main = "Fisher's Iris Data")
par(op)

## 'x' a "Date" vector :
nd <- length(dv <- seq(as.Date("1959-02-21"), by = "weeks", length.out = 100))
mSC <- cbind(I=1, sin=sin(pi*(1:nd)/8), cos=cos(pi*(1:nd)/8))
matplot(dv, mSC, type = "b", main = "matplot(<Date>, y)")

## 'x' a "POSIXct" date-time vector :
ct <- seq(c(ISOdate(2000,3,20)), by = "15 mins", length.out = 100)
matplot(ct, mSC, type = "b", main = "matplot(<POSIXct>, y)")
## or the same with even more axis flexibility:
matplot(ct, mSC, type = "b", main = "matplot(<POSIXct>, y)", xaxt="n")
Axis(ct, side=1, at = ct[1+4*(0:24)])

## Also works for data frame columns:
matplot(iris[1:50,1:4])
```

`mosaicplot` *Mosaic Plots*

Description

Plots a mosaic on the current graphics device.

Usage

```
mosaicplot(x, ...)

## Default S3 method:
mosaicplot(x, main = deparse1(substitute(x)),
           sub = NULL, xlab = NULL, ylab = NULL,
           sort = NULL, off = NULL, dir = NULL,
           color = NULL, shade = FALSE, margin = NULL,
           cex.axis = 0.66, las = par("las"), border = NULL,
           type = c("pearson", "deviance", "FT"), ...)

## S3 method for class 'formula'
mosaicplot(formula, data = NULL, ...,
           main = deparse1(substitute(data)), subset,
           na.action = stats::na.omit)
```

Arguments

x	a contingency table in array form, with optional category labels specified in the dimnames(x) attribute. The table is best created by the table() command.
main	character string for the mosaic title.
sub	character string for the mosaic sub-title (at bottom).
xlab, ylab	x- and y-axis labels used for the plot; by default, the first and second element of names(dimnames(X)) (i.e., the name of the first and second variable in X).
sort	vector ordering of the variables, containing a permutation of the integers 1:length(dim(x)) (the default).
off	vector of offsets to determine percentage spacing at each level of the mosaic (appropriate values are between 0 and 20, and the default is 20 times the number of splits for 2-dimensional tables, and 10 otherwise). Rescaled to maximally 50, and recycled if necessary.
dir	vector of split directions ("v" for vertical and "h" for horizontal) for each level of the mosaic, one direction for each dimension of the contingency table. The default consists of alternating directions, beginning with a vertical split.
color	logical or (recycling) vector of colors for color shading, used only when shade is FALSE, or NULL (default). By default, grey boxes are drawn. color = TRUE uses grey.colors for a gamma-corrected grey palette. color = FALSE gives empty boxes with no shading.
shade	a logical indicating whether to produce extended mosaic plots, or a numeric vector of at most 5 distinct positive numbers giving the absolute values of the cut points for the residuals. By default, shade is FALSE, and simple mosaics are created. Using shade = TRUE cuts absolute values at 2 and 4.
margin	a list of vectors with the marginal totals to be fit in the log-linear model. By default, an independence model is fitted. See loglin for further information.
cex.axis	The magnification to be used for axis annotation, as a multiple of par("cex").
las	numeric; the style of axis labels, see par.
border	colour of borders of cells: see polygon.
type	a character string indicating the type of residual to be represented. Must be one of "pearson" (giving components of Pearson's χ^2), "deviance" (giving components of the likelihood ratio χ^2), or "FT" for the Freeman-Tukey residuals. The value of this argument can be abbreviated.
formula	a formula, such as y ~ x.
data	a data frame (or list), or a contingency table from which the variables in formula should be taken.
...	further arguments to be passed to or from methods.
subset	an optional vector specifying a subset of observations in the data frame to be used for plotting.
na.action	a function which indicates what should happen when the data contains variables to be cross-tabulated, and these variables contain NAs. The default is to omit cases which have an NA in any variable. Since the tabulation will omit all cases containing missing values, this will only be useful if the na.action function replaces missing values.

Details

This is a generic function. It currently has a default method (`mosaicplot.default`) and a formula interface (`mosaicplot.formula`).

Extended mosaic displays visualize standardized residuals of a loglinear model for the table by color and outline of the mosaic's tiles. (Standardized residuals are often referred to a standard normal distribution.) Cells representing negative residuals are drawn in shaded of red and with broken borders; positive ones are drawn in blue with solid borders.

For the formula method, if `data` is an object inheriting from class `"table"` or class `"ftable"` or an array with more than 2 dimensions, it is taken as a contingency table, and hence all entries should be non-negative. In this case the left-hand side of `formula` should be empty and the variables on the right-hand side should be taken from the names of the dimnames attribute of the contingency table. A marginal table of these variables is computed, and a mosaic plot of that table is produced.

Otherwise, `data` should be a data frame or matrix, list or environment containing the variables to be cross-tabulated. In this case, after possibly selecting a subset of the data as specified by the `subset` argument, a contingency table is computed from the variables given in `formula`, and a mosaic is produced from this.

See Emerson (1998) for more information and a case study with television viewer data from Nielsen Media Research.

Missing values are not supported except via an `na.action` function when `data` contains variables to be cross-tabulated.

A more flexible and extensible implementation of mosaic plots written in the grid graphics system is provided in the function `mosaic` in the contributed package **vcd** (Meyer, Zeileis and Hornik, 2006).

Author(s)

S-PLUS original by John Emerson <`john.emerson@yale.edu`>. Originally modified and enhanced for R by Kurt Hornik.

References

Hartigan, J.A., and Kleiner, B. (1984). A mosaic of television ratings. *The American Statistician*, **38**, 32–35. doi:10.2307/2683556.

Emerson, J. W. (1998). Mosaic displays in S-PLUS: A general implementation and a case study. *Statistical Computing and Graphics Newsletter (ASA)*, **9**, 1, 17–23.

Friendly, M. (1994). Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, **89**, 190–200. doi:10.2307/2291215.

Meyer, D., Zeileis, A., and Hornik, K. (2006) The strucplot Framework: Visualizing Multi-Way Contingency Tables with **vcd**. *Journal of Statistical Software*, **17(3)**, 1–48. doi:10.18637/jss.v017.i03.

See Also

`assocplot`, `loglin`.

Examples

```
require(stats)
mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)
## Formula interface for tabulated data:
mosaicplot(~ Sex + Age + Survived, data = Titanic, color = TRUE)
```

```
mosaicplot(HairEyeColor, shade = TRUE)
## Independence model of hair and eye color and sex.  Indicates that
## there are more blue eyed blonde females than expected in the case
## of independence and too few brown eyed blonde females.
## The corresponding model is:
fm <- loglin(HairEyeColor, list(1, 2, 3))
pchisq(fm$pearson, fm$df, lower.tail = FALSE)

mosaicplot(HairEyeColor, shade = TRUE, margin = list(1:2, 3))
## Model of joint independence of sex from hair and eye color.  Males
## are underrepresented among people with brown hair and eyes, and are
## overrepresented among people with brown hair and blue eyes.
## The corresponding model is:
fm <- loglin(HairEyeColor, list(1:2, 3))
pchisq(fm$pearson, fm$df, lower.tail = FALSE)

## Formula interface for raw data: visualize cross-tabulation of numbers
## of gears and carburettors in Motor Trend car data.
mosaicplot(~ gear + carb, data = mtcars, color = TRUE, las = 1)
# color recycling
mosaicplot(~ gear + carb, data = mtcars, color = 2:3, las = 1)
```

`mtext` *Write Text into the Margins of a Plot*

Description

Text is written in one of the four margins of the current figure region or one of the outer margins of the device region.

Usage

```
mtext(text, side = 3, line = 0, outer = FALSE, at = NA,
      adj = NA, padj = NA, cex = NA, col = NA, font = NA, ...)
```

Arguments

`text`	a character or expression vector specifying the *text* to be written. Other objects are coerced by `as.graphicsAnnot`.
`side`	on which side of the plot (1=bottom, 2=left, 3=top, 4=right).
`line`	on which MARgin line, starting at 0 counting outwards.
`outer`	use outer margins if available.
`at`	give location of each string in user coordinates. If the component of `at` corresponding to a particular text item is not a finite value (the default), the location will be determined by `adj`.
`adj`	adjustment for each string in reading direction. For strings parallel to the axes, `adj = 0` means left or bottom alignment, and `adj = 1` means right or top alignment. If `adj` is not a finite value (the default), the value of `par("las")` determines the adjustment. For strings plotted parallel to the axis the default is to centre the string.

padj	adjustment for each string perpendicular to the reading direction (which is controlled by adj). For strings parallel to the axes, padj = 0 means bottom alignment, and padj = 1 means top alignment (relative to the axis line). If padj is not a finite value (the default), the value of par("las") determines the adjustment. For strings plotted perpendicular to the axis the default is to centre the string.
cex	character expansion factor. NULL and NA are equivalent to 1.0. This is an absolute measure, not scaled by par("cex") or by setting par("mfrow") or par("mfcol"). Can be a vector.
col	color to use. Can be a vector. NA values (the default) mean use par("col").
font	font for text. Can be a vector. NA values (the default) mean use par("font").
...	Further graphical parameters (see par), including family, las and xpd. (The latter defaults to the figure region unless outer = TRUE, otherwise the device region. It can only be increased.)

Details

The user coordinates in the outer margins always range from zero to one, and are not affected by the user coordinates in the figure region(s) — R differs here from other implementations of S.

All of the named arguments can be vectors, and recycling will take place to plot as many strings as the longest of the vector arguments.

Note that a vector adj has a different meaning from text. adj = 0.5 will centre the string, but for outer = TRUE on the device region rather than the plot region.

Parameter las will determine the orientation of the string(s). For strings plotted perpendicular to the axis the default justification is to place the end of the string nearest the axis on the specified line. (Note that this differs from S, which uses srt if at is supplied and las if it is not. Parameter srt is ignored in R.)

Note that if the text is to be plotted perpendicular to the axis, adj determines the justification of the string *and* the position along the axis unless at is specified.

Graphics parameter "ylbias" (see par) determines how the text baseline is placed relative to the nominal line.

Side Effects

The given text is written onto the current plot.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

title, text, plot, par; plotmath for details on mathematical annotation.

Examples

```
plot(1:10, (-4:5)^2, main = "Parabola Points", xlab = "xlab")
mtext("10 of them")
for(s in 1:4)
    mtext(paste("mtext(..., line= -1, {side, col, font} = ", s,
```

```
          ", cex = ", (1+s)/2, ")"), line = -1,
          side = s, col = s, font = s, cex = (1+s)/2)
mtext("mtext(..., line= -2)", line = -2)
mtext("mtext(..., line= -2, adj = 0)", line = -2, adj = 0)
##--- log axis :
plot(1:10, exp(1:10), log = "y", main = "log =\"y\"", xlab = "xlab")
for(s in 1:4) mtext(paste("mtext(...,side=", s ,")"), side = s)

##--- illustrating padj behavior :
plot(0, axes=FALSE, ann=FALSE, frame.plot=TRUE)
for(si in 1:4) mtext(c("padj=0", "-----------", "padj=1"),
                     side = si, padj = c(0, 0.5, 1))
```

`pairs` *Scatterplot Matrices*

Description

A matrix of scatterplots is produced.

Usage

```
pairs(x, ...)

## S3 method for class 'formula'
pairs(formula, data = NULL, ..., subset,
      na.action = stats::na.pass)

## Default S3 method:
pairs(x, labels, panel = points, ...,
      horInd = 1:nc, verInd = 1:nc,
      lower.panel = panel, upper.panel = panel,
      diag.panel = NULL, text.panel = textPanel,
      label.pos = 0.5 + has.diag/3, line.main = 3,
      cex.labels = NULL, font.labels = 1,
      row1attop = TRUE, gap = 1, log = "",
      horOdd = !row1attop, verOdd = !row1attop)
```

Arguments

`x`	the coordinates of points given as numeric columns of a matrix or data frame. Logical and factor columns are converted to numeric in the same way that `data.matrix` does.
`formula`	a formula, such as `~ x + y + z`. Each term will give a separate variable in the pairs plot, so terms should be numeric vectors. (A response will be interpreted as another variable, but not treated specially, so it is confusing to use one.)
`data`	a data.frame (or list) from which the variables in `formula` should be taken.
`subset`	an optional vector specifying a subset of observations to be used for plotting.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is to pass missing values on to the panel functions, but `na.action = na.omit` will cause cases with missing values in any of the variables to be omitted entirely.

labels	the names of the variables.
panel	function(x, y, ...) which is used to plot the contents of each panel of the display.
...	arguments to be passed to or from methods. Also, graphical parameters can be given as can arguments to plot such as main. par("oma") will be set appropriately unless specified.
horInd, verInd	The (numerical) indices of the variables to be plotted on the horizontal and vertical axes respectively.
lower.panel, upper.panel	separate panel functions (or NULL) to be used below and above the diagonal respectively.
diag.panel	optional function(x, ...) to be applied on the diagonals.
text.panel	optional function(x, y, labels, cex, font, ...) to be applied on the diagonals.
label.pos	y position of labels in the text panel.
line.main	if main is specified, line.main gives the line argument to mtext() which draws the title. You may want to specify oma when changing line.main.
cex.labels, font.labels	graphics parameters for the text panel.
row1attop	logical. Should the layout be matrix-like with row 1 at the top, or graph-like with row 1 at the bottom? The latter (non default) leads to a basically symmetric scatterplot matrix.
gap	distance between subplots, in margin lines.
log	a character string indicating if logarithmic axes are to be used, see plot.default *or* a numeric vector of indices specifying the indices of those variables where logarithmic axes should be used for both x and y. log = "xy" specifies logarithmic axes for all variables.
horOdd, verOdd	logical (or integer) determining how the horizontal and vertical axis labeling happens. If true, the axis labelling starts at the first (from top left) row or column, respectively.

Details

The ij-th scatterplot contains x[,i] plotted against x[,j]. The scatterplot can be customised by setting panel functions to appear as something completely different. The off-diagonal panel functions are passed the appropriate columns of x as x and y: the diagonal panel function (if any) is passed a single column, and the text.panel function is passed a single (x, y) location and the column name. Setting some of these panel functions to NULL is equivalent to *not* drawing anything there.

The graphical parameters pch and col can be used to specify a vector of plotting symbols and colors to be used in the plots.

The graphical parameter oma will be set by pairs.default unless supplied as an argument.

A panel function should not attempt to start a new plot, but just plot within a given coordinate system: thus plot and boxplot are not panel functions.

By default, missing values are passed to the panel functions and will often be ignored within a panel. However, for the formula method and na.action = na.omit, all cases which contain a missing values for any of the variables are omitted completely (including when the scales are selected).

Arguments horInd and verInd were introduced in R 3.2.0. If given the same value they can be used to select or re-order variables: with different ranges of consecutive values they can be used to plot rectangular windows of a full pairs plot; in the latter case 'diagonal' refers to the diagonal of the full plot.

Author(s)

Enhancements for R 1.0.0 contributed by Dr. Jens Oehlschlägel-Akiyoshi and R-core members.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```
pairs(iris[1:4], main = "Anderson's Iris Data -- 3 species",
      pch = 21, bg = c("red", "green3", "blue")[unclass(iris$Species)])

## formula method, "graph" layout (row 1 at bottom):
pairs(~ Fertility + Education + Catholic, data = swiss, row1attop=FALSE,
      subset = Education < 20, main = "Swiss data, Education < 20")

pairs(USJudgeRatings, gap=1/10) # (gap: not wasting plotting area)
## show only lower triangle (and suppress labeling for whatever reason):
pairs(USJudgeRatings, text.panel = NULL, upper.panel = NULL)

## put histograms on the diagonal
panel.hist <- function(x, ...)
{
    usr <- par("usr")
    par(usr = c(usr[1:2], 0, 1.5) )
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}
pairs(USJudgeRatings[1:5], panel = panel.smooth,
      cex = 1.5, pch = 24, bg = "light blue", horOdd=TRUE,
      diag.panel = panel.hist, cex.labels = 2, font.labels = 2)

## put (absolute) correlations on the upper panels,
## with size proportional to the correlations.
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
    par(usr = c(0, 1, 0, 1))
    r <- abs(cor(x, y))
    txt <- format(c(r, 0.123456789), digits = digits)[1]
    txt <- paste0(prefix, txt)
    if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex = cex.cor * r)
}
pairs(USJudgeRatings, lower.panel = panel.smooth, upper.panel = panel.cor,
      gap=0, row1attop=FALSE)

pairs(iris[-5], log = "xy") # plot all variables on log scale
pairs(iris, log = 1:4, # log the first four
```

```
main = "Lengths and Widths in [log]", line.main=1.5, oma=c(2,2,3,2))
```

`panel.smooth` *Simple Panel Plot*

Description

An example of a simple useful `panel` function to be used as argument in e.g., `coplot` or `pairs`.

Usage

```
panel.smooth(x, y, col = par("col"), bg = NA, pch = par("pch"),
             cex = 1, col.smooth = 2, span = 2/3, iter = 3,
             ...)
```

Arguments

`x`, `y`	numeric vectors of the same length
`col`, `bg`, `pch`, `cex`	numeric or character codes for the color(s), point type and size of `points`; see also `par`.
`col.smooth`	color to be used by `lines` for drawing the smooths.
`span`	smoothing parameter `f` for `lowess`, see there.
`iter`	number of robustness iterations for `lowess`.
`...`	further arguments to `lines`.

See Also

`coplot` and `pairs` where `panel.smooth` is typically used; `lowess` which does the smoothing.

Examples

```
pairs(swiss, panel = panel.smooth, pch = ".")  # emphasize the smooths
pairs(swiss, panel = panel.smooth, lwd = 2, cex = 1.5, col = 4)  # hmm...
```

`par` *Set or Query Graphical Parameters*

Description

`par` can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Usage

```
par(..., no.readonly = FALSE)

<highlevel plot> (...., <tag> = <value>)
```

Arguments

`...`	arguments in `tag = value` form, a single list of tagged values, or character vectors of parameter names. Supported parameters are described in the 'Graphical Parameters' section.
`no.readonly`	logical; if `TRUE` and there are no other arguments, only parameters are returned which can be set by a subsequent `par()` call *on the same device*.

Details

Each device has its own set of graphical parameters. If the current device is the null device, `par` will open a new device before querying/setting parameters. (What device is controlled by `options("device")`.)

Parameters are queried by giving one or more character vectors of parameter names to `par`.

`par()` (no arguments) or `par(no.readonly = TRUE)` is used to get *all* the graphical parameters (as a named list). Their names are currently taken from the unexported variable `graphics:::.Pars`.

R.O. indicates ***read-only arguments***: These may only be used in queries and cannot be set. (`"cin"`, `"cra"`, `"csi"`, `"cxy"`, `"din"` and `"page"` are always read-only.)

Several parameters can only be set by a call to `par()`:

- `"ask"`,
- `"fig"`, `"fin"`,
- `"lheight"`,
- `"mai"`, `"mar"`, `"mex"`, `"mfcol"`, `"mfrow"`, `"mfg"`,
- `"new"`,
- `"oma"`, `"omd"`, `"omi"`,
- `"pin"`, `"plt"`, `"ps"`, `"pty"`,
- `"usr"`,
- `"xlog"`, `"ylog"`,
- `"ylbias"`

The remaining parameters can also be set as arguments (often via `...`) to high-level plot functions such as `plot.default`, `plot.window`, `points`, `lines`, `abline`, `axis`, `title`, `text`, `mtext`, `segments`, `symbols`, `arrows`, `polygon`, `rect`, `box`, `contour`, `filled.contour` and `image`. Such settings will be active during the execution of the function, only. However, see the comments on `bg`, `cex`, `col`, `lty`, `lwd` and `pch` which may be taken as *arguments* to certain plot functions rather than as graphical parameters.

The meaning of 'character size' is not well-defined: this is set up for the device taking `pointsize` into account but often not the actual font family in use. Internally the corresponding pars (`cra`, `cin`, `cxy` and `csi`) are used only to set the inter-line spacing used to convert `mar` and `oma` to physical margins. (The same inter-line spacing multiplied by `lheight` is used for multi-line strings in `text` and `strheight`.)

Note that graphical parameters are suggestions: plotting functions and devices need not make use of them (and this is particularly true of non-default methods for e.g. `plot`).

Value

When parameters are set, their previous values are returned in an invisible named list. Such a list can be passed as an argument to `par` to restore the parameter values. Use `par(no.readonly = TRUE)` for the full list of parameters that can be restored. However, restoring all of these is not wise: see the 'Note' section.

When just one parameter is queried, the value of that parameter is returned as (atomic) vector. When two or more parameters are queried, their values are returned in a list, with the list names giving the parameters.

Note the inconsistency: setting one parameter returns a list, but querying one parameter returns a vector.

Graphical Parameters

`adj` The value of `adj` determines the way in which text strings are justified in `text`, `mtext` and `title`. 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. Note that whereas for `text` it refers to positioning of text about a point, for `mtext` and `title` it controls placement within the plot or device region.

`ann` If set to `FALSE`, high-level plotting functions calling `plot.default` do not annotate the plots they produce with axis titles and overall titles. The default is to do annotation.

`ask` logical. If `TRUE` (and the R session is interactive) the user is asked for input, before a new figure is drawn. As this applies to the device, it also affects output by packages **grid** and **lattice**. It can be set even on non-screen devices but may have no effect there.

This not really a graphics parameter, and its use is deprecated in favour of `devAskNewPage`.

`bg` The color to be used for the background of the device region. When called from `par()` it also sets `new = FALSE`. See section 'Color Specification' for suitable values. For many devices the initial value is set from the `bg` argument of the device, and for the rest it is normally `"white"`.

Note that some graphics functions such as `plot.default` and `points` have an *argument* of this name with a different meaning.

`bty` A character string which determined the type of `box` which is drawn about plots. If `bty` is one of `"o"` (the default), `"l"`, `"7"`, `"c"`, `"u"`, or `"]"` the resulting box resembles the corresponding upper case letter. A value of `"n"` suppresses the box.

`cex` A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. This starts as `1` when a device is opened, and is reset when the layout is changed, e.g. by setting `mfrow`.

Note that some graphics functions such as `plot.default` have an *argument* of this name which *multiplies* this graphical parameter, and some functions such as `points` and `text` accept a vector of values which are recycled.

`cex.axis` The magnification to be used for axis annotation relative to the current setting of `cex`.

`cex.lab` The magnification to be used for x and y labels relative to the current setting of `cex`.

`cex.main` The magnification to be used for main titles relative to the current setting of `cex`.

`cex.sub` The magnification to be used for sub-titles relative to the current setting of `cex`.

`cin` ***R.O.***; character size `(width, height)` in inches. These are the same measurements as `cra`, expressed in different units.

`col` A specification for the default plotting color. See section 'Color Specification'.
Some functions such as `lines` and `text` accept a vector of values which are recycled and may be interpreted slightly differently.

`col.axis` The color to be used for axis annotation. Defaults to `"black"`.

`col.lab` The color to be used for x and y labels. Defaults to `"black"`.

`col.main` The color to be used for plot main titles. Defaults to `"black"`.

`col.sub` The color to be used for plot sub-titles. Defaults to `"black"`.

`cra` ***R.O.***; size of default character `(width, height)` in 'rasters' (pixels). Some devices have no concept of pixels and so assume an arbitrary pixel size, usually 1/72 inch. These are the same measurements as `cin`, expressed in different units.

`crt` A numerical value specifying (in degrees) how single characters should be rotated. It is unwise to expect values other than multiples of 90 to work. Compare with `srt` which does string rotation.

`csi` ***R.O.***; height of (default-sized) characters in inches. The same as `par("cin")[2]`.

`cxy` ***R.O.***; size of default character `(width, height)` in user coordinate units. `par("cxy")` is `par("cin")/par("pin")` scaled to user coordinates. Note that `c(strwidth(ch), strheight(ch))` for a given string `ch` is usually much more precise.

`din` ***R.O.***; the device dimensions, `(width, height)`, in inches. See also `dev.size`, which is updated immediately when an on-screen device windows is re-sized.

`err` (*Unimplemented*; R is silent when points outside the plot region are *not* plotted.) The degree of error reporting desired.

`family` The name of a font family for drawing text. The maximum allowed length is 200 bytes. This name gets mapped by each graphics device to a device-specific font description. The default value is `""` which means that the default device fonts will be used (and what those are should be listed on the help page for the device). Standard values are `"serif"`, `"sans"` and `"mono"`, and the Hershey font families are also available. (Devices may define others, and some devices will ignore this setting completely. Names starting with `"Hershey"` are treated specially and should only be used for the built-in Hershey font families.) This can be specified inline for `text`.

`fg` The color to be used for the foreground of plots. This is the default color used for things like axes and boxes around plots. When called from `par()` this also sets parameter `col` to the same value. See section 'Color Specification'. A few devices have an argument to set the initial value, which is otherwise `"black"`.

`fig` A numerical vector of the form `c(x1, x2, y1, y2)` which gives the (NDC) coordinates of the figure region in the display region of the device. If you set this, unlike S, you start a new plot, so to add to an existing plot use `new = TRUE` as well.

`fin` The figure region dimensions, `(width, height)`, in inches. If you set this, unlike S, you start a new plot.

`font` An integer which specifies which font to use for text. If possible, device drivers arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in Adobe symbol encoding. On some devices font families can be selected by `family` to choose different sets of 5 fonts.

`font.axis` The font to be used for axis annotation.

`font.lab` The font to be used for x and y labels.

`font.main` The font to be used for plot main titles.

`font.sub` The font to be used for plot sub-titles.

`lab` A numerical vector of the form `c(x, y, len)` which modifies the default way that axes are annotated. The values of `x` and `y` give the (approximate) number of tickmarks on the x and y axes and `len` specifies the label length. The default is `c(5, 5, 7)`. `len` *is unimplemented* in R.

`las` numeric in {0,1,2,3}; the style of axis labels.

0: always parallel to the axis [*default*],

1: always horizontal,

2: always perpendicular to the axis,

3: always vertical.

Also supported by `mtext`. Note that string/character rotation *via* argument `srt` to `par` does *not* affect the axis labels.

`lend` The line end style. This can be specified as an integer or string:

`0` and `"round"` mean rounded line caps [*default*];

`1` and `"butt"` mean butt line caps;

`2` and `"square"` mean square line caps.

`lheight` The line height multiplier. The height of a line of text (used to vertically space multi-line text) is found by multiplying the character height both by the current character expansion and by the line height multiplier. Default value is 1. Used in `text` and `strheight`.

`ljoin` The line join style. This can be specified as an integer or string:

`0` and `"round"` mean rounded line joins [*default*];

`1` and `"mitre"` mean mitred line joins;

`2` and `"bevel"` mean bevelled line joins.

`lmitre` The line mitre limit. This controls when mitred line joins are automatically converted into bevelled line joins. The value must be larger than 1 and the default is 10. Not all devices will honour this setting.

`lty` The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, or `"twodash"`, where `"blank"` uses 'invisible lines' (i.e., does not draw them).

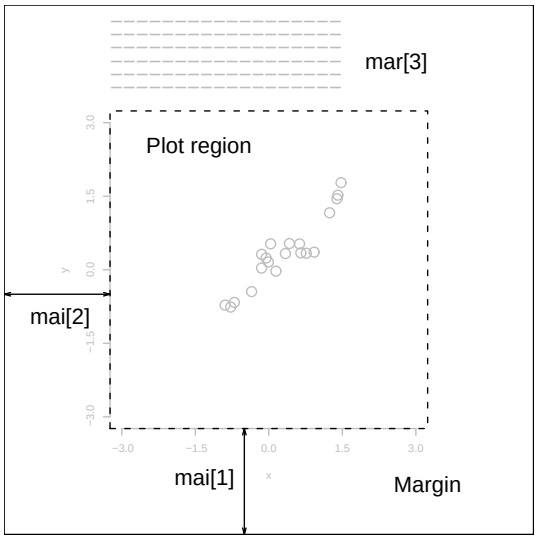
Alternatively, a string of up to 8 characters (from `c(1:9, "A":"F")`) may be given, giving the length of line segments which are alternatively drawn and skipped. See section 'Line Type Specification'.

Functions such as `lines` and `segments` accept a vector of values which are recycled.

`lwd` The line width, a *positive* number, defaulting to `1`. The interpretation is device-specific, and some devices do not implement line widths less than one. (See the help on the device for details of the interpretation.)

Functions such as `lines` and `segments` accept a vector of values which are recycled: in such uses lines corresponding to values `NA` or `NaN` are omitted. The interpretation of `0` is device-specific.

`mai` A numerical vector of the form `c(bottom, left, top, right)` which gives the margin size specified in inches.



`mar` A numerical vector of the form `c(bottom, left, top, right)` which gives the number of lines of margin to be specified on the four sides of the plot. The default is `c(5, 4, 4, 2) + 0.1`.

`mex` `mex` is a character size expansion factor which is used to describe coordinates in the margins of plots. Note that this does not change the font size, rather specifies the size of font (as a multiple of `csi`) used to convert between `mar` and `mai`, and between `oma` and `omi`.

This starts as `1` when the device is opened, and is reset when the layout is changed (alongside resetting `cex`).

`mfcol, mfrow` A vector of the form `c(nr, nc)`. Subsequent figures will be drawn in an `nr`-by-`nc` array on the device by *columns* (`mfcol`), or *rows* (`mfrow`), respectively.

In a layout with exactly two rows and columns the base value of `"cex"` is reduced by a factor of 0.83: if there are three or more of either rows or columns, the reduction factor is 0.66.

Setting a layout resets the base value of `cex` and that of `mex` to `1`.

If either of these is queried it will give the current layout, so querying cannot tell you the order in which the array will be filled.

Consider the alternatives, `layout` and `split.screen`.

`mfg` A numerical vector of the form `c(i, j)` where `i` and `j` indicate which figure in an array of figures is to be drawn next (if setting) or is being drawn (if enquiring). The array must already have been set by `mfcol` or `mfrow`.

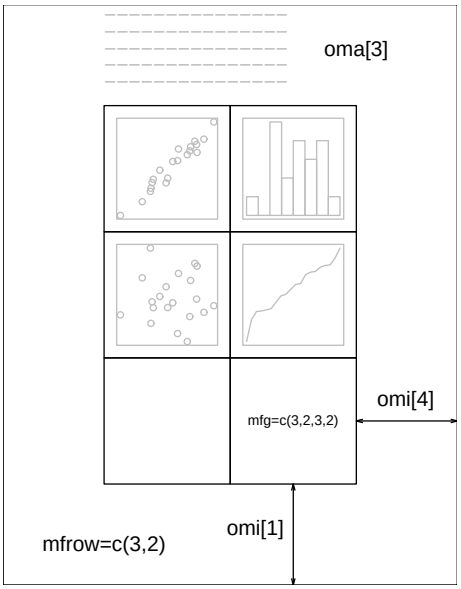
For compatibility with S, the form `c(i, j, nr, nc)` is also accepted, when `nr` and `nc` should be the current number of rows and number of columns. Mismatches will be ignored, with a warning.

`mgp` The margin line (in `mex` units) for the axis title, axis labels and axis line. Note that `mgp[1]` affects `title` whereas `mgp[2:3]` affect `axis`. The default is `c(3, 1, 0)`.

`mkh` The height in inches of symbols to be drawn when the value of `pch` is an integer. *Completely ignored in* R.

`new` logical, defaulting to `FALSE`. If set to `TRUE`, the next high-level plotting command (actually `plot.new`) should *not clean* the frame before drawing *as if it were on a* **new** *device*. It is an error (ignored with a warning) to try to use `new = TRUE` on a device that does not currently contain a high-level plot.

`oma` A vector of the form `c(bottom, left, top, right)` giving the size of the outer margins in lines of text.



`omd` A vector of the form `c(x1, x2, y1, y2)` giving the region *inside* outer margins in NDC (= normalized device coordinates), i.e., as a fraction (in $[0, 1]$) of the device region.

`omi` A vector of the form `c(bottom, left, top, right)` giving the size of the outer margins in inches.

`page` ***R.O.***; A boolean value indicating whether the next call to `plot.new` is going to start a new page. This value may be `FALSE` if there are multiple figures on the page.

`pch` Either an integer specifying a symbol or a single character to be used as the default in plotting points. See `points` for possible values and their interpretation. Note that only integers and single-character strings can be set as a graphics parameter (and not `NA` nor `NULL`).

Some functions such as `points` accept a vector of values which are recycled.

`pin` The current plot dimensions, `(width, height)`, in inches.

`plt` A vector of the form `c(x1, x2, y1, y2)` giving the coordinates of the plot region as fractions of the current figure region.

`ps` integer; the point size of text (but not symbols). Unlike the `pointsize` argument of most devices, this does not change the relationship between `mar` and `mai` (nor `oma` and `omi`).

What is meant by 'point size' is device-specific, but most devices mean a multiple of 1bp, that is 1/72 of an inch.

`pty` A character specifying the type of plot region to be used; `"s"` generates a square plotting region and `"m"` generates the maximal plotting region.

`smo` (*Unimplemented*) a value which indicates how smooth circles and circular arcs should be.

`srt` The string rotation in degrees. See the comment about `crt`. Only supported by `text`.

`tck` The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If `tck >= 0.5` it is interpreted as a fraction of the relevant side, so if `tck = 1` grid lines are drawn. The default setting (`tck = NA`) is to use `tcl = -0.5`.

`tcl` The length of tick marks as a fraction of the height of a line of text. The default value is `-0.5`; setting `tcl = NA` sets `tck = -0.01` which is S' default.

`usr` A vector of the form `c(x1, x2, y1, y2)` giving the extremes of the user coordinates of the plotting region. When a logarithmic scale is in use (i.e., `par("xlog")` is true, see below), then the x-limits will be `10 ^ par("usr")[1:2]`. Similarly for the y-axis.

`xaxp` A vector of the form `c(x1, x2, n)` giving the coordinates of the extreme tick marks and the number of intervals between tick-marks when `par("xlog")` is false. Otherwise, when *log* coordinates are active, the three values have a different meaning: For a small range, `n` is *negative*, and the ticks are as in the linear case, otherwise, `n` is in `1:3`, specifying a case number, and `x1` and `x2` are the lowest and highest power of 10 inside the user coordinates, `10 ^ par("usr")[1:2]`. (The `"usr"` coordinates are log10-transformed here!)

n = 1 will produce tick marks at 10^j for integer j,

n = 2 gives marks $k10^j$ with $k \in \{1, 5\}$,

n = 3 gives marks $k10^j$ with $k \in \{1, 2, 5\}$.

See `axTicks()` for a pure R implementation of this.

This parameter is reset when a user coordinate system is set up, for example by starting a new page or by calling `plot.window` or setting `par("usr")`: `n` is taken from `par("lab")`. It affects the default behaviour of subsequent calls to `axis` for sides 1 or 3.

It is only relevant to default numeric axis systems, and not for example to dates.

`xaxs` The style of axis interval calculation to be used for the x-axis. Possible values are `"r"`, `"i"`, `"e"`, `"s"`, `"d"`. The styles are generally controlled by the range of data or `xlim`, if given.
Style `"r"` (regular) first extends the data range by 4 percent at each end and then finds an axis with pretty labels that fits within the extended range.
Style `"i"` (internal) just finds an axis with pretty labels that fits within the original data range.
Style `"s"` (standard) finds an axis with pretty labels within which the original data range fits.
Style `"e"` (extended) is like style `"s"`, except that it is also ensures that there is room for plotting symbols within the bounding box.
Style `"d"` (direct) specifies that the current axis should be used on subsequent plots.
(*Only* `"r"` *and* `"i"` *styles have been implemented in* R.)

`xaxt` A character which specifies the x axis type. Specifying `"n"` suppresses plotting of the axis. The standard value is `"s"`: for compatibility with S values `"l"` and `"t"` are accepted but are equivalent to `"s"`: any value other than `"n"` implies plotting.

`xlog` A logical value (see `log` in `plot.default`). If `TRUE`, a logarithmic scale is in use (e.g., after `plot(*, log = "x")`). For a new device, it defaults to `FALSE`, i.e., linear scale.

`xpd` A logical value or `NA`. If `FALSE`, all plotting is clipped to the plot region, if `TRUE`, all plotting is clipped to the figure region, and if `NA`, all plotting is clipped to the device region. See also `clip`.

`yaxp` A vector of the form `c(y1, y2, n)` giving the coordinates of the extreme tick marks and the number of intervals between tick-marks unless for log coordinates, see `xaxp` above.

`yaxs` The style of axis interval calculation to be used for the y-axis. See `xaxs` above.

`yaxt` A character which specifies the y axis type. Specifying `"n"` suppresses plotting.

`ylbias` A positive real value used in the positioning of text in the margins by `axis` and `mtext`. The default is in principle device-specific, but currently `0.2` for all of R's own devices. Set this to `0.2` for compatibility with R < 2.14.0 on `x11` and `windows()` devices.

`ylog` A logical value; see `xlog` above.

Color Specification

Colors can be specified in several different ways. The simplest way is with a character string giving the color name (e.g., `"red"`). A list of the possible colors can be obtained with the function `colors`. Alternatively, colors can be specified directly in terms of their RGB components with a string of the form `"#RRGGBB"` where each of the pairs `RR`, `GG`, `BB` consist of two hexadecimal digits giving a value in the range `00` to `FF`. Hexadecimal colors can be in the long hexadecimal form (e.g., `"#rrggbb"` or `"#rrggbbaa"`) or the short form (e.g, `"#rgb"` or `"#rgba"`). The short form is expanded to the long

form by replicating digits (not by adding zeroes), e.g., "#rgb" becomes "#rrggbb". Colors can also be specified by giving an index into a small table of colors, the palette: indices wrap round so with the default palette of size 8, 10 is the same as 2. This provides compatibility with S. Index 0 corresponds to the background color. Note that the palette (apart from 0 which is per-device) is a per-session setting.

Negative integer colours are errors.

Additionally, "transparent" is *transparent*, useful for filled areas (such as the background!), and just invisible for things like lines or text. In most circumstances (integer) NA is equivalent to "transparent" (but not for text and mtext).

Semi-transparent colors are available for use on devices that support them.

The functions rgb, hsv, hcl, gray and rainbow provide additional ways of generating colors.

Line Type Specification

Line types can either be specified by giving an index into a small built-in table of line types (1 = solid, 2 = dashed, etc, see lty above) or directly as the lengths of on/off stretches of line. This is done with a string of an even number (up to eight) of characters, namely *non-zero* (hexadecimal) digits which give the lengths in consecutive positions in the string. For example, the string "33" specifies three units on followed by three off and "3313" specifies three units on followed by three off followed by one on and finally three off. The 'units' here are (on most devices) proportional to lwd, and with lwd = 1 are in pixels or points or 1/96 inch.

The five standard dash-dot line types (lty = 2:6) correspond to c("44", "13", "1343", "73", "2262").

Note that NA is not a valid value for lty.

Note

The effect of restoring all the (settable) graphics parameters as in the examples is hard to predict if the device has been resized. Several of them are attempting to set the same things in different ways, and those last in the alphabet will win. In particular, the settings of mai, mar, pin, plt and pty interact, as do the outer margin settings, the figure layout and figure region size.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

plot.default for some high-level plotting parameters; colors; clip; options for other setup parameters; graphic devices x11, pdf, postscript and setting up device regions by layout and split.screen.

Examples

```
op <- par(mfrow = c(2, 2), # 2 x 2 pictures on one plot
          pty = "s")       # square plotting region,
                           # independent of device size

## At end of plotting, reset to previous settings:
par(op)
```

```
## Alternatively,
op <- par(no.readonly = TRUE) # the whole list of settable par's.
## do lots of plotting and par(.) calls, then reset:
par(op)
## Note this is not in general good practice

par("ylog") # FALSE
plot(1 : 12, log = "y")
par("ylog") # TRUE

plot(1:2, xaxs = "i") # 'inner axis' w/o extra space
par(c("usr", "xaxp"))

( nr.prof <-
c(prof.pilots = 16, lawyers = 11, farmers = 10, salesmen = 9, physicians = 9,
  mechanics = 6, policemen = 6, managers = 6, engineers = 5, teachers = 4,
  housewives = 3, students = 3, armed.forces = 1))
par(las = 3)
barplot(rbind(nr.prof)) # R 0.63.2: shows alignment problem
par(las = 0)  # reset to default

require(grDevices) # for gray
## 'fg' use:
plot(1:12, type = "b", main = "'fg' : axes, ticks and box in gray",
     fg = gray(0.7), bty = "7" , sub = R.version.string)

ex <- function() {
   old.par <- par(no.readonly = TRUE) # all par settings which
                                      # could be changed.
   on.exit(par(old.par))
   ## ...
   ## ... do lots of par() settings and plots
   ## ...
   invisible() #-- now,  par(old.par)  will be executed
}
ex()

## Line types
showLty <- function(ltys, xoff = 0, ...) {
   stopifnot((n <- length(ltys)) >= 1)
   op <- par(mar = rep(.5,4)); on.exit(par(op))
   plot(0:1, 0:1, type = "n", axes = FALSE, ann = FALSE)
   y <- (n:1)/(n+1)
   clty <- as.character(ltys)
   mytext <- function(x, y, txt)
      text(x, y, txt, adj = c(0, -.3), cex = 0.8, ...)
   abline(h = y, lty = ltys, ...); mytext(xoff, y, clty)
   y <- y - 1/(3*(n+1))
   abline(h = y, lty = ltys, lwd = 2, ...)
   mytext(1/8+xoff, y, paste(clty," lwd = 2"))
}
showLty(c("solid", "dashed", "dotted", "dotdash", "longdash", "twodash"))
par(new = TRUE)  # the same:
showLty(c("solid", "44", "13", "1343", "73", "2262"), xoff = .2, col = 2)
showLty(c("11", "22", "33", "44",   "12", "13", "14",   "21", "31"))
```

persp *Perspective Plots*

Description

This function draws perspective plots of a surface over the x–y plane. `persp` is a generic function.

Usage

```
persp(x, ...)

## Default S3 method:
persp(x = seq(0, 1, length.out = nrow(z)),
      y = seq(0, 1, length.out = ncol(z)),
      z, xlim = range(x), ylim = range(y),
      zlim = range(z, na.rm = TRUE),
      xlab = NULL, ylab = NULL, zlab = NULL,
      main = NULL, sub = NULL,
      theta = 0, phi = 15, r = sqrt(3), d = 1,
      scale = TRUE, expand = 1,
      col = "white", border = NULL, ltheta = -135, lphi = 0,
      shade = NA, box = TRUE, axes = TRUE, nticks = 5,
      ticktype = "simple", ...)
```

Arguments

x, y	locations of grid lines at which the values in z are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively.
z	a matrix containing the values to be plotted (NAs are allowed). Note that x can be used instead of z for convenience.
xlim, ylim, zlim	x-, y- and z-limits. These should be chosen to cover the range of values of the surface: see 'Details'.
xlab, ylab, zlab	titles for the axes. N.B. These must be character strings; expressions are not accepted. Numbers will be coerced to character strings.
main, sub	main title and subtitle, as for title.
theta, phi	angles defining the viewing direction. theta gives the azimuthal direction and phi the colatitude.
r	the distance of the eyepoint from the centre of the plotting box.
d	a value which can be used to vary the strength of the perspective transformation. Values of d greater than 1 will lessen the perspective effect and values less and 1 will exaggerate it.
scale	before viewing the x, y and z coordinates of the points defining the surface are transformed to the interval [0,1]. If scale is TRUE the x, y and z coordinates are transformed separately. If scale is FALSE the coordinates are scaled so that aspect ratios are retained. This is useful for rendering things like DEM information.
expand	a expansion factor applied to the z coordinates. Often used with 0 < expand < 1 to shrink the plotting box in the z direction.

col	the color(s) of the surface facets. Transparent colours are ignored. This is recycled to the $(nx-1)(ny-1)$ facets.
border	the color of the line drawn around the surface facets. The default, NULL, corresponds to par("fg"). A value of NA will disable the drawing of borders: this is sometimes useful when the surface is shaded.
ltheta, lphi	if finite values are specified for ltheta and lphi, the surface is shaded as though it was being illuminated from the direction specified by azimuth ltheta and colatitude lphi.
shade	the shade at a surface facet is computed as ((1+d)/2)^shade, where d is the dot product of a unit vector normal to the facet and a unit vector in the direction of a light source. Values of shade close to one yield shading similar to a point light source model and values close to zero produce no shading. Values in the range 0.5 to 0.75 provide an approximation to daylight illumination.
box	should the bounding box for the surface be displayed. The default is TRUE.
axes	should ticks and labels be added to the box. The default is TRUE. If box is FALSE then no ticks or labels are drawn.
ticktype	character: "simple" draws just an arrow parallel to the axis to indicate direction of increase; "detailed" draws normal ticks as per 2D plots.
nticks	the (approximate) number of tick marks to draw on the axes. Has no effect if ticktype is "simple".
...	additional graphical parameters (see par).

Details

The plots are produced by first transforming the (x,y,z) coordinates to the interval [0,1] using the limits supplied or computed from the range of the data. The surface is then viewed by looking at the origin from a direction defined by theta and phi. If theta and phi are both zero the viewing direction is directly down the negative y axis. Changing theta will vary the azimuth and changing phi the colatitude.

There is a hook called "persp" (see setHook) called after the plot is completed, which is used in the testing code to annotate the plot page. The hook function(s) are called with no argument.

Notice that persp interprets the z matrix as a table of f(x[i], y[j]) values, so that the x axis corresponds to row number and the y axis to column number, with column 1 at the bottom, so that with the standard rotation angles, the top left corner of the matrix is displayed at the left hand side, closest to the user.

The sizes and fonts of the axis labels and the annotations for ticktype = "detailed" are controlled by graphics parameters "cex.lab"/"font.lab" and "cex.axis"/"font.axis" respectively.

The bounding box is drawn with edges of faces facing away from the viewer (and hence at the back of the box) with solid lines and other edges dashed and on top of the surface. This (and the plotting of the axes) assumes that the axis limits are chosen so that the surface is within the box, and the function will warn if this is not the case.

Value

persp() returns the *viewing transformation matrix*, say VT, a 4×4 matrix suitable for projecting 3D coordinates (x, y, z) into the 2D plane using homogeneous 4D coordinates (x, y, z, t). It can be used to superimpose additional graphical elements on the 3D plot, by lines() or points(), using the function trans3d().

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`contour` and `image`; `trans3d`.

Rotatable 3D plots can be produced by package **rgl**: other ways to produce static perspective plots are available in packages **lattice** and **scatterplot3d**.

Examples

```
require(grDevices) # for trans3d
## More examples in  demo(persp) !!
##                   -----------

# (1) The Obligatory Mathematical surface.
#     Rotated sinc function.

x <- seq(-10, 10, length.out = 30)
y <- x
f <- function(x, y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
op <- par(bg = "white")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
      ltheta = 120, shade = 0.75, ticktype = "detailed",
      xlab = "X", ylab = "Y", zlab = "Sinc( r )", cex.axis = 0.8
) -> res
round(res, 3)

# (2) Add to existing persp plot - using trans3d() :

xE <- c(-10,10); xy <- expand.grid(xE, xE)
points(trans3d(xy[,1], xy[,2], z = 6,          pmat = res), col = 2, pch = 16)
lines (trans3d(x,      y = 10, z = 6 + sin(x), pmat = res), col = 3)

phi <- seq(0, 2*pi, length.out = 201)
r1 <- 7.725 # radius of 2nd maximum
xr <- r1 * cos(phi)
yr <- r1 * sin(phi)
lines(trans3d(xr,yr, f(xr,yr), res), col = "pink", lwd = 2)
## (no hidden lines)

# (3) Visualizing a simple DEM model

z <- 2 * volcano        # Exaggerate the relief
x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
## Don't draw the grid lines :  border = NA
par(bg = "slategray")
persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
      ltheta = -120, shade = 0.75, border = NA, box = FALSE)

# (4) Surface colours corresponding to z-values
```

```
par(bg = "white")
x <- seq(-1.95, 1.95, length.out = 30)
y <- seq(-1.95, 1.95, length.out = 35)
z <- outer(x, y, function(a, b) a*b^2)
nrz <- nrow(z)
ncz <- ncol(z)
# Create a function interpolating colors in the range of specified colors
jet.colors <- colorRampPalette( c("blue", "green") )
# Generate the desired number of colors from this palette
nbcol <- 100
color <- jet.colors(nbcol)
# Compute the z-value at the facet centres
zfacet <- z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
# Recode facet z-values into color indices
facetcol <- cut(zfacet, nbcol)
persp(x, y, z, col = color[facetcol], phi = 30, theta = -30)

par(op)
```

`pie` *Pie Charts*

Description

Draw a pie chart.

Usage

```
pie(x, labels = names(x), edges = 200, radius = 0.8,
    clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
    density = NULL, angle = 45, col = NULL, border = NULL,
    lty = NULL, main = NULL, ...)
```

Arguments

`x`	a vector of non-negative numerical quantities. The values in `x` are displayed as the areas of pie slices.
`labels`	one or more expressions or character strings giving names for the slices. Other objects are coerced by `as.graphicsAnnot`. For empty or `NA` (after coercion to character) labels, no label nor pointing line is drawn.
`edges`	the circular outline of the pie is approximated by a polygon with this many edges.
`radius`	the pie is drawn centered in a square box whose sides range from -1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.
`clockwise`	logical indicating if slices are drawn clockwise or counter clockwise (i.e., mathematically positive direction), the latter is default.
`init.angle`	number specifying the *starting angle* (in degrees) for the slices. Defaults to 0 (i.e., '3 o'clock') unless `clockwise` is true where `init.angle` defaults to 90 (degrees), (i.e., '12 o'clock').

`density`	the density of shading lines, in lines per inch. The default value of `NULL` means that no shading lines are drawn. Non-positive values of `density` also inhibit the drawing of shading lines.
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise).
`col`	a vector of colors to be used in filling or shading the slices. If missing a set of 6 pastel colours is used, unless `density` is specified when `par("fg")` is used.
`border, lty`	(possibly vectors) arguments passed to `polygon` which draws each slice.
`main`	an overall title for the plot.
`...`	graphical parameters can be given as arguments to `pie`. They will affect the main title and labels only.

Note

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

Cleveland (1985), page 264: "Data that can be shown by pie charts always can be shown by a dot chart. This means that judgements of position along a common scale can be made instead of the less accurate angle judgements." This statement is based on the empirical investigations of Cleveland and McGill as well as investigations by perceptual psychologists.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Cleveland, W. S. (1985) *The Elements of Graphing Data*. Wadsworth: Monterey, CA, USA.

See Also

`dotchart`.

Examples

```
require(grDevices)
pie(rep(1, 24), col = rainbow(24), radius = 0.9)

pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(pie.sales) <- c("Blueberry", "Cherry",
    "Apple", "Boston Cream", "Other", "Vanilla Cream")
pie(pie.sales) # default colours
pie(pie.sales, col = c("purple", "violetred1", "green3",
                       "cornsilk", "cyan", "white"))
pie(pie.sales, col = gray(seq(0.4, 1.0, length.out = 6)))
pie(pie.sales, density = 10, angle = 15 + 10 * 1:6)
pie(pie.sales, clockwise = TRUE, main = "pie(*, clockwise = TRUE)")
segments(0, 0, 0, 1, col = "red", lwd = 2)
text(0, 1, "init.angle = 90", col = "red")

n <- 200
pie(rep(1, n), labels = "", col = rainbow(n), border = NA,
    main = "pie(*, labels=\"\", col=rainbow(n), border=NA,..")

## Another case showing pie() is rather fun than science:
```

```
## (original by FinalBackwardsGlance on http://imgur.com/gallery/wWrpU4X)
pie(c(Sky = 78, "Sunny side of pyramid" = 17, "Shady side of pyramid" = 5),
    init.angle = 315, col = c("deepskyblue", "yellow", "yellow3"), border = FALSE)
```

`plot.data.frame` *Plot Method for Data Frames*

Description

`plot.data.frame`, a method for the `plot` generic. It is designed for a quick look at numeric data frames.

Usage

```
## S3 method for class 'data.frame'
plot(x, ...)
```

Arguments

`x`	object of class `data.frame`.
`...`	further arguments to `stripchart`, `plot.default` or `pairs`.

Details

This is intended for data frames with *numeric* columns. For more than two columns it first calls `data.matrix` to convert the data frame to a numeric matrix and then calls `pairs` to produce a scatterplot matrix. This can fail and may well be inappropriate: for example numerical conversion of dates will lose their special meaning and a warning will be given.

For a two-column data frame it plots the second column against the first by the most appropriate method for the first column.

For a single numeric column it uses `stripchart`, and for other single-column data frames tries to find a plot method for the single column.

See Also

`data.frame`

Examples

```
plot(OrchardSprays[1], method = "jitter")
plot(OrchardSprays[c(4,1)])
plot(OrchardSprays)

plot(iris)
plot(iris[5:4])
plot(women)
```

plot.default *The Default Scatterplot Function*

Description

Draw a scatter plot with decorations such as axes and titles in the active graphics window.

Usage

```
## Default S3 method:
plot(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA,
     xgap.axis = NA, ygap.axis = NA,
     ...)
```

Arguments

x, y	the x and y arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function xy.coords for details. If supplied separately, they must be of the same length.
type	1-character string giving the type of plot desired. The following values are possible, for details, see plot: "p" for points, "l" for lines, "b" for both points and lines, "c" for empty points joined by lines, "o" for overplotted points and lines, "s" and "S" for stair steps and "h" for histogram-like vertical lines. Finally, "n" does not produce any points or lines.
xlim	the x limits (x1, x2) of the plot. Note that x1 > x2 is allowed and leads to a 'reversed axis'. The default value, NULL, indicates that the range of the finite values to be plotted should be used.
ylim	the y limits of the plot.
log	a character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.
main	a main title for the plot, see also title.
sub	a subtitle for the plot.
xlab	a label for the x axis, defaults to a description of x.
ylab	a label for the y axis, defaults to a description of y.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
axes	a logical value indicating whether both axes should be drawn on the plot. Use graphical parameter "xaxt" or "yaxt" to suppress just one of the axes.
frame.plot	a logical indicating whether a box should be drawn around the plot.
panel.first	an 'expression' to be evaluated after the plot axes are set up but before any plotting takes place. This can be useful for drawing background grids or scatterplot smooths. Note that this works by lazy evaluation: passing this argument from other plot methods may well not work since it may be evaluated too early.

`panel.last` an expression to be evaluated after plotting has taken place but before the axes, title and box are added. See the comments about `panel.first`.

`asp` the y/x aspect ratio, see `plot.window`.

`xgap.axis`, `ygap.axis` the x/y axis gap factors, passed as `gap.axis` to the two `axis()` calls (when `axes` is true, as per default).

`...` other graphical parameters (see `par` and section 'Details' below).

Details

Commonly used graphical parameters are:

`col` The colors for lines and points. Multiple colors can be specified so that each point can be given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines will all be plotted in the first colour specified.

`bg` a vector of background colors for open plot symbols, see `points`. Note: this is **not** the same setting as `par("bg")`.

`pch` a vector of plotting characters or symbols: see `points`.

`cex` a numerical vector giving the amount by which plotting characters and symbols should be scaled relative to the default. This works as a multiple of `par("cex")`. `NULL` and `NA` are equivalent to `1.0`. Note that this does not affect annotation: see below.

`lty` a vector of line types, see `par`.

`cex.main`**,** `col.lab`**,** `font.sub`**, etc** settings for main- and sub-title and axis annotation, see `title` and `par`.

`lwd` a vector of line widths, see `par`.

Note

The presence of `panel.first` and `panel.last` is a historical anomaly: default plots do not have 'panels', unlike e.g. `pairs` plots. For more control, use lower-level plotting functions: `plot.default` calls in turn some of `plot.new`, `plot.window`, `plot.xy`, `axis`, `box` and `title`, and plots can be built up by calling these individually, or by calling `plot(type = "n")` and adding further elements.

The `plot` generic was moved from the **graphics** package to the **base** package in R 4.0.0. It is currently re-exported from the **graphics** namespace to allow packages importing it from there to continue working, but this may change in future versions of R.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Cleveland, W. S. (1985) *The Elements of Graphing Data.* Monterey, CA: Wadsworth.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

`plot`, `plot.window`, `xy.coords`. For thousands of points, consider using `smoothScatter` instead.

Examples

```
Speed <- cars$speed
Distance <- cars$dist
plot(Speed, Distance, panel.first = grid(8, 8),
     pch = 0, cex = 1.2, col = "blue")
plot(Speed, Distance,
     panel.first = lines(stats::lowess(Speed, Distance), lty = "dashed"),
     pch = 0, cex = 1.2, col = "blue")

## Show the different plot types
x <- 0:12
y <- sin(pi/5 * x)
op <- par(mfrow = c(3,3), mar = .1+ c(2,2,3,1))
for (tp in c("p","l","b",  "c","o","h",  "s","S","n")) {
   plot(y ~ x, type = tp, main = paste0("plot(*, type = \"", tp, "\")"))
   if(tp == "S") {
      lines(x, y, type = "s", col = "red", lty = 2)
      mtext("lines(*, type = \"s\", ...)", col = "red", cex = 0.8)
   }
}
par(op)

##--- Log-Log Plot  with  custom axes
lx <- seq(1, 5, length.out = 41)
yl <- expression(e^{-frac(1,2) * {log[10](x)}^2})
y <- exp(-.5*lx^2)
op <- par(mfrow = c(2,1), mar = par("mar")-c(1,0,2,0), mgp = c(2, .7, 0))
plot(10^lx, y, log = "xy", type = "l", col = "purple",
     main = "Log-Log plot", ylab = yl, xlab = "x")
plot(10^lx, y, log = "xy", type = "o", pch = ".", col = "forestgreen",
     main = "Log-Log plot with custom axes", ylab = yl, xlab = "x",
     axes = FALSE, frame.plot = TRUE)
my.at <- 10^(1:5)
axis(1, at = my.at, labels = formatC(my.at, format = "fg"))
e.y <- -5:-1 ; at.y <- 10^e.y
axis(2, at = at.y, col.axis = "red", las = 1,
     labels = as.expression(lapply(e.y, function(E) bquote(10^.(E)))))
par(op)
```

`plot.design` *Plot Univariate Effects of a Design or Model*

Description

Plot univariate effects of one or more `factor`s, typically for a designed experiment as analyzed by `aov()`.

Usage

```
plot.design(x, y = NULL, fun = mean, data = NULL, ...,
            ylim = NULL, xlab = "Factors", ylab = NULL,
            main = NULL, ask = NULL, xaxt = par("xaxt"),
            axes = TRUE, xtick = FALSE)
```

Arguments

x	either a data frame containing the design factors and optionally the response, or a formula or terms object.
y	the response, if not given in x.
fun	a function (or name of one) to be applied to each subset. It must return one number for a numeric (vector) input.
data	data frame containing the variables referenced by x when that is formula-like.
...	graphical parameters such as col, see par.
ylim	range of y values, as in plot.default.
xlab	x axis label, see title.
ylab	y axis label with a 'smart' default.
main	main title, see title.
ask	logical indicating if the user should be asked before a new page is started – in the case of multiple y values.
xaxt	character giving the type of x axis.
axes	logical indicating if axes should be drawn.
xtick	logical indicating if ticks (one per factor) should be drawn on the x axis.

Details

The supplied function will be called once for each level of each factor in the design and the plot will show these summary values. The levels of a particular factor are shown along a vertical line, and the overall value of fun() for the response is drawn as a horizontal line.

Note

A big effort was taken to make this closely compatible to the S version. However, col (and fg) specifications have different effects.

In S this was a method of the plot generic function for design objects.

Author(s)

Roberto Frisullo and Martin Maechler

References

Chambers, J. M. and Hastie, T. J. eds (1992) *Statistical Models in S*. Chapman & Hall, London, **the** *white book*, pp. 546–7 (and 163–4).

Freeny, A. E. and Landwehr, J. M. (1990) Displays for data from large designed experiments; Computer Science and Statistics: Proc.\ 22nd Symp\. Interface, 117–126, Springer Verlag.

See Also

interaction.plot for a 'standard graphic' of designed experiments.

Examples

```
require(stats)
plot.design(warpbreaks)  # automatic for data frame with one numeric var.

Form <- breaks ~ wool + tension
summary(fm1 <- aov(Form, data = warpbreaks))
plot.design(       Form, data = warpbreaks, col = 2)  # same as above

## More than one y :
utils::str(esoph)
plot.design(esoph) ## two plots; if interactive you are "ask"ed

## or rather, compare mean and median:
op <- par(mfcol = 1:2)
plot.design(ncases/ncontrols ~ ., data = esoph, ylim = c(0, 0.8))
plot.design(ncases/ncontrols ~ ., data = esoph, ylim = c(0, 0.8),
            fun = median)
par(op)
```

`plot.factor` *Plotting Factor Variables*

Description

This functions implements a scatterplot method for `factor` arguments of the *generic* `plot` function.

If `y` is missing `barplot` is produced. For numeric `y` a `boxplot` is used, and for a factor `y` a `spineplot` is shown. For any other type of `y` the next `plot` method is called, normally `plot.default`.

Usage

```
## S3 method for class 'factor'
plot(x, y, legend.text = NULL, ...)
```

Arguments

`x, y`	numeric or factor. `y` may be missing.
`legend.text`	character vector for annotation of `y` axis in the case of a factor `y`: defaults to `levels(y)`. This sets the `yaxlabels` argument of `spineplot`.
`...`	Further arguments to `barplot`, `boxplot`, `spineplot` or `plot` as appropriate. All of these accept graphical parameters (see `par`) and annotation arguments passed to `title` and `axes = FALSE`. None accept `type`.

See Also

`plot.default`, `plot.formula`, `barplot`, `boxplot`, `spineplot`.

Examples

```
require(grDevices)
plot(weight ~ group, data = PlantGrowth)           # numeric vector ~ factor
plot(cut(weight, 2) ~ group, data = PlantGrowth)   # factor ~ factor
## passing "..." to spineplot() eventually:
plot(cut(weight, 3) ~ group, data = PlantGrowth,
     col = hcl(c(0, 120, 240), 50, 70))

plot(PlantGrowth$group, axes = FALSE, main = "no axes")  # extremely silly
```

---

plot.formula *Formula Notation for Scatterplots*

Description

Specify a scatterplot or add points, lines, or text via a formula.

Usage

```
## S3 method for class 'formula'
plot(formula, data = parent.frame(), ..., subset,
             ylab = varnames[response], ask = dev.interactive())

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

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

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

Arguments

formula	a formula, such as y ~ x.
data	a data.frame (or list) from which the variables in formula should be taken. A matrix is converted to a data frame.
...	Arguments to be passed to or from other methods. horizontal = TRUE is also accepted.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
ylab	the y label of the plot(s).
ask	logical, see par.

Details

For the lines, points and text methods the formula should be of the form y ~ x or y ~ 1 with a left-hand side and a single term on the right-hand side. The plot method accepts other forms discussed later in this section.

Both the terms in the formula and the ... arguments are evaluated in data enclosed in parent.frame() if data is a list or a data frame. The terms of the formula and those arguments in ... that are of the same length as data are subjected to the subsetting specified in subset. A plot against the running index can be specified as plot(y ~ 1).

If the formula in the plot method contains more than one term on the right-hand side, a series of plots is produced of the response against each non-response term.

For the plot method the formula can be of the form ~ z + y + z: the variables specified on the right-hand side are collected into a data frame, subsetted if specified, and displayed by plot.data.frame.

Missing values are not considered in these methods, and in particular cases with missing values are not removed.

If y is an object (i.e., has a class attribute) then plot.formula looks for a plot method for that class first. Otherwise, the class of x will determine the type of the plot. For factors this will be a parallel boxplot, and argument horizontal = TRUE can be specified (see boxplot).

Note that some arguments will need to be protected from premature evaluation by enclosing them in quote: currently this is done automatically for main, sub and xlab. For example, it is needed for the panel.first and panel.last arguments passed to plot.default.

Value

These functions are invoked for their side effect of drawing on the active graphics device.

See Also

plot.default, points, lines, plot.factor.

Examples

```
op <- par(mfrow = c(2,1))
plot(Ozone ~ Wind, data = airquality, pch = as.character(Month))
plot(Ozone ~ Wind, data = airquality, pch = as.character(Month),
     subset = Month != 7)
par(op)

## text.formula() can be very natural:
wb <- within(warpbreaks, {
    time <- seq_along(breaks); W.T <- wool:tension })
plot(breaks ~ time, data = wb, type = "b")
text(breaks ~ time, data = wb, labels = W.T, col = 1+as.integer(wool))
```

---

plot.histogram *Plot Histograms*

Description

Plotting methods for objects of class "histogram", typically produced by hist.

Usage

```
## S3 method for class 'histogram'
plot(x, freq = equidist, density = NULL, angle = 45,
               col = "lightgray", border = NULL, lty = NULL,
               main = paste("Histogram of", paste(x$xname, collapse = "\n")),
               sub = NULL, xlab = x$xname, ylab,
               xlim = range(x$breaks), ylim = NULL,
               axes = TRUE, labels = FALSE, add = FALSE,
               ann = TRUE, ...)

## S3 method for class 'histogram'
lines(x, ...)
```

Arguments

x	a histogram object, or a list with components density, mid, etc, see hist for information about the components of x.
freq	logical; if TRUE, the histogram graphic is to present a representation of frequencies, i.e, x$counts; if FALSE, *relative* frequencies (probabilities), i.e., x$density, are plotted. The default is true for equidistant breaks and false otherwise.
col	a colour to be used to fill the bars. The default has been changed from NULL (unfilled bars) only as from R 4.2.0.
border	the color of the border around the bars.
angle, density	select shading of bars by lines: see rect.
lty	the line type used for the bars, see also lines.
main, sub, xlab, ylab	these arguments to title have useful defaults here.
xlim, ylim	the range of x and y values with sensible defaults.
axes	logical, indicating if axes should be drawn.
labels	logical or character. Additionally draw labels on top of bars, if not FALSE; if TRUE, draw the counts or rounded densities; if labels is a character, draw itself.
add	logical. If TRUE, only the bars are added to the current plot. This is what lines.histogram(*) does.
ann	logical. Should annotations (titles and axis titles) be plotted?
...	further graphical parameters to title and axis.

Details

lines.histogram(*) is the same as plot.histogram(*, add = TRUE).

See Also

hist, stem, density.

Examples

```
(wwt <- hist(women$weight, nclass = 7, plot = FALSE))
plot(wwt, labels = TRUE) # default main & xlab using wwt$xname
plot(wwt, border = "dark blue", col = "light blue",
     main = "Histogram of 15 women's weights", xlab = "weight [pounds]")

## Fake "lines" example, using non-default labels:
w2 <- wwt; w2$counts <- w2$counts - 1
lines(w2, col = "Midnight Blue", labels = ifelse(w2$counts, "> 1", "1"))
```

---

plot.raster *Plotting Raster Images*

Description

This functions implements a plot method for raster images.

Usage

```
## S3 method for class 'raster'
plot(x, y,
     xlim = c(0, ncol(x)), ylim = c(0, nrow(x)),
     xaxs = "i", yaxs = "i",
     asp = 1, add = FALSE, ...)
```

Arguments

x, y	raster. y will be ignored.
xlim, ylim	Limits on the plot region (default from dimensions of the raster).
xaxs, yaxs	Axis interval calculation style (default means that raster fills plot region).
asp	Aspect ratio (default retains aspect ratio of the raster).
add	Logical indicating whether to simply add raster to an existing plot.
...	Further arguments to the rasterImage function.

See Also

plot.default, rasterImage.

Examples

```
require(grDevices)
r <- as.raster(c(0.5, 1, 0.5))
plot(r)
# additional arguments to rasterImage()
plot(r, interpolate=FALSE)
# distort
plot(r, asp=NA)
# fill page
op <- par(mar=rep(0, 4))
plot(r, asp=NA)
par(op)
```

```
# normal annotations work
plot(r, asp=NA)
box()
title(main="This is my raster")
# add to existing plot
plot(1)
plot(r, add=TRUE)
```

`plot.table` *Plot Methods for* `table` *Objects*

Description

This is a method of the generic `plot` function for (contingency) `table` objects. Whereas for two- and more dimensional tables, a `mosaicplot` is drawn, one-dimensional ones are plotted as bars.

Usage

```
## S3 method for class 'table'
plot(x, type = "h", ylim = c(0, max(x)), lwd = 2,
     xlab = NULL, ylab = NULL, frame.plot = is.num, ...)
## S3 method for class 'table'
points(x, y = NULL, type = "h", lwd = 2, ...)
## S3 method for class 'table'
lines(x, y = NULL, type = "h", lwd = 2, ...)
```

Arguments

`x`	a `table` (like) object.
`y`	Must be `NULL`: there to protect against incorrect calls.
`type`	plotting type.
`ylim`	range of y-axis.
`lwd`	line width for bars when `type = "h"` is used in the 1D case.
`xlab, ylab`	x- and y-axis labels.
`frame.plot`	logical indicating if a frame (`box`) should be drawn in the 1D case. Defaults to true when `x` has `dimnames` coerce-able to numbers.
`...`	further graphical arguments, see `plot.default`. `axes = FALSE` is accepted.

See Also

`plot.factor`, the `plot` method for factors.

Examples

```
## 1-d tables
(Poiss.tab <- table(N = stats::rpois(200, lambda = 5)))
plot(Poiss.tab, main = "plot(table(rpois(200, lambda = 5)))")

plot(table(state.division))

## 4-D :
plot(Titanic, main ="plot(Titanic, main= *)")
```

plot.window *Set up World Coordinates for Graphics Window*

Description

This function sets up the world coordinate system for a graphics window. It is called by higher level functions such as `plot.default` (*after* `plot.new`).

Usage

```
plot.window(xlim, ylim, log = "", asp = NA, ...)
```

Arguments

`xlim`, `ylim`	numeric vectors of length 2, giving the x and y coordinates ranges.
`log`	character; indicating which axes should be in log scale.
`asp`	numeric, giving the **asp**ect ratio y/x, see 'Details'.
`...`	further graphical parameters as in `par`. The relevant ones are `xaxs`, `yaxs` and `lab`.

Details

asp: If `asp` is a finite positive value then the window is set up so that one data unit in the y direction is equal in length to `asp` $\times$ one data unit in the x direction.

Note that in this case, `par("usr")` is no longer determined by, e.g., `par("xaxs")`, but rather by `asp` and the device's aspect ratio. (See what happens if you interactively resize the plot device after running the example below!)

The special case `asp == 1` produces plots where distances between points are represented accurately on screen. Values with `asp > 1` can be used to produce more accurate maps when using latitude and longitude.

Note that the coordinate ranges will be extended by 4% if the appropriate graphical parameter `xaxs` or `yaxs` has value `"r"` (which is the default).

To reverse an axis, use `xlim` or `ylim` of the form `c(hi, lo)`.

The function attempts to produce a plausible set of scales if one or both of `xlim` and `ylim` is of length one or the two values given are identical, but it is better to avoid that case.

Usually, one should rather use the higher-level functions such as `plot`, `hist`, `image`, ..., instead and refer to their help pages for explanation of the arguments.

A side-effect of the call is to set up the `usr`, `xaxp` and `yaxp` graphical parameters. (It is for the latter two that `lab` is used.)

See Also

`xy.coords`, `plot.xy`, `plot.default`.

`par` for the graphical parameters mentioned.

Examples

```
##--- An example for the use of 'asp' :
require(stats)  # normally loaded
loc <- cmdscale(eurodist)
rx <- range(x <- loc[,1])
ry <- range(y <- -loc[,2])
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "")
abline(h = pretty(rx, 10), v = pretty(ry, 10), col = "lightgray")
text(x, y, labels(eurodist), cex = 0.8)
```

---

plot.xy *Basic Internal Plot Function*

Description

This is *the* internal function that does the basic plotting of points and lines. Usually, one should rather use the higher level functions instead and refer to their help pages for explanation of the arguments.

Usage

```
plot.xy(xy, type, pch = par("pch"), lty = par("lty"),
        col = par("col"), bg = NA,
        cex = 1, lwd = par("lwd"), ...)
```

Arguments

xy	A four-element list as results from xy.coords.
type	1 character code: see plot.default. NULL is accepted as a synonym for "p".
pch	character or integer code for kind of points, see points.default.
lty	line type code, see lines.
col	color code or name, see colors, palette. Here NULL means colour 0.
bg	background (fill) color for the open plot symbols 21:25: see points.default.
cex	character expansion.
lwd	line width, also used for (non-filled) plot symbols, see lines and points.
...	further graphical parameters such as xpd, lend, ljoin and lmitre.

Details

The arguments pch, col, bg, cex, lwd may be vectors and may be recycled, depending on type: see points and lines for specifics. In particular note that lwd is treated as a vector for points and as a single (first) value for lines.

cex is a numeric factor in addition to par("cex") which affects symbols and characters as drawn by type "p", "o", "b" and "c".

See Also

plot, plot.default, points, lines.

Examples

```
points.default # to see how it calls "plot.xy(xy.coords(x, y), ...)"
```

`points` *Add Points to a Plot*

Description

`points` is a generic function to draw a sequence of points at the specified coordinates. The specified character(s) are plotted, centered at the coordinates.

Usage

```
points(x, ...)

## Default S3 method:
points(x, y = NULL, type = "p", ...)
```

Arguments

`x, y`	coordinate vectors of points to plot.
`type`	character indicating the type of plotting; actually any of the `types` as in `plot.default`.
`...`	Further graphical parameters may also be supplied as arguments. See 'Details'.

Details

The coordinates can be passed in a plotting structure (a list with `x` and `y` components), a two-column matrix, a time series, ... See `xy.coords`. If supplied separately, they must be of the same length.

Graphical parameters commonly used are

`pch` plotting 'character', i.e., symbol to use. This can either be a single character or an integer code for one of a set of graphics symbols. The full set of S symbols is available with `pch = 0:18`, see the examples below. (NB: R uses circles instead of the octagons used in S.)

Value `pch = "."` (equivalently `pch = 46`) is handled specially. It is a rectangle of side 0.01 inch (scaled by `cex`). In addition, if `cex = 1` (the default), each side is at least one pixel (1/72 inch on the `pdf`, `postscript` and `xfig` devices).

For other text symbols, `cex = 1` corresponds to the default font size of the device, often specified by an argument `pointsize`. For `pch` in `0:25` the default size is about 75% of the character height (see `par("cin")`).

`col` color code or name, see `par`.

`bg` background (fill) color for the open plot symbols given by `pch = 21:25`.

`cex` character (or symbol) expansion: a numerical vector. This works as a multiple of `par("cex")`.

`lwd` line width for drawing symbols see `par`.

Others less commonly used are `lty` and `lwd` for types such as `"b"` and `"l"`.

The graphical parameters `pch`, `col`, `bg`, `cex` and `lwd` can be vectors (which will be recycled as needed) giving a value for each point plotted. If lines are to be plotted (e.g., for `type = "b"`) the first element of `lwd` is used.

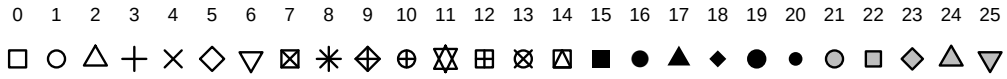
Points whose `x`, `y`, `pch`, `col` or `cex` value is `NA` are omitted from the plot.

`pch` values

Values of `pch` are stored internally as integers. The interpretation is

- `NA_integer_`: no symbol.
- `0:18`: S-compatible vector symbols.
- `19:25`: further R vector symbols.
- `26:31`: unused (and ignored).
- `32:127`: ASCII characters.
- `128:255` native characters *only in a single-byte locale and for the symbol font*. (`128:159` are only used on Windows.)
- `-32 ...` Unicode code point (where supported).

Note that unlike S (which uses octagons), symbols `1`, `10`, `13` and `16` use circles. The filled shapes `15:18` do not include a border.



The following R plotting symbols are can be obtained with `pch = 19:25`: those with `21:25` can be colored and filled with different colors: `col` gives the border color and `bg` the background color (which is '`"grey"`' in the figure)

- `pch = 19`: solid circle,
- `pch = 20`: bullet (smaller solid circle, 2/3 the size of `19`),
- `pch = 21`: filled circle,
- `pch = 22`: filled square,
- `pch = 23`: filled diamond,
- `pch = 24`: filled triangle point-up,
- `pch = 25`: filled triangle point down.

Note that all of these both fill the shape and draw a border. Some care in interpretation is needed when semi-transparent colours are used for both fill and border (and the result might be device-specific and even viewer-specific for `pdf`).

The difference between `pch = 16` and `pch = 19` is that the latter uses a border and so is perceptibly larger when `lwd` is large relative to `cex`.

Values `pch = 26:31` are currently unused and `pch = 32:127` give the ASCII characters. In a single-byte locale `pch = 128:255` give the corresponding character (if any) in the locale's character set. Where supported by the OS, negative values specify a Unicode code point, so e.g. `-0x2642L` is a 'male sign' and `-0x20ACL` is the Euro.

A character string consisting of a single character is converted to an integer: `32:127` for ASCII characters, and usually to the Unicode code point otherwise. (In non-Latin-1 single-byte locales, `128:255` will be used for 8-bit characters.)

If `pch` supplied is a logical, integer or character `NA` or an empty character string the point is omitted from the plot.

If `pch` is `NULL` or otherwise of length 0, `par("pch")` is used.

If the symbol font (`par(font = 5)`) is used, numerical values should be used for `pch`: the range is `c(32:126, 160:254)` in all locales (but `240` is not defined (used for 'apple' on macOS) and `160`, Euro, may not be present).

Note

A single-byte encoding may include the characters in `pch = 128:255`, and if it does, a font may not include all (or even any) of them.

Not all negative numbers are valid as Unicode code points, and no check is done. A display device is likely to use a rectangle for (or omit) Unicode code points which are invalid or for which it does not have a glyph in the font used.

What happens for very small or zero values of `cex` is device-dependent: symbols or characters may become invisible or they may be plotted at a fixed minimum size. Circles of zero radius will not be plotted.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`points.formula` for the formula method; `plot`, `lines`, and the underlying workhorse function `plot.xy`.

Examples

```
require(stats) # for rnorm
plot(-4:4, -4:4, type = "n")  # setting up coord. system
points(rnorm(200), rnorm(200), col = "red")
points(rnorm(100)/2, rnorm(100)/2, col = "blue", cex = 1.5)

op <- par(bg = "light blue")
x <- seq(0, 2*pi, length.out = 51)
## something "between type='b' and type='o'":
plot(x, sin(x), type = "o", pch = 21, bg = par("bg"), col = "blue", cex = .6,
 main = 'plot(..., type="o", pch=21, bg=par("bg"))')
par(op)

## Illustration of pch = 0:25 (as in the figure shown above in PDF/HTML help)
## Not run: png("pch.png", height = 0.7, width = 7, res = 100, units = "in")
par(mar = rep(0,4))
plot(c(-1, 26), 0:1, type = "n", axes = FALSE)
text(0:25, 0.6, 0:25, cex = 0.5)
points(0:25, rep(0.3, 26), pch = 0:25, bg = "grey")


##-------- Showing all the extra & some char graphics symbols ---------
pchShow <-
  function(extras = c("*",".", "o","O","0","+","-","|","%","#"),
           cex = 3, ## good for both .Device=="postscript" and "x11"
           col = "red3", bg = "gold", coltext = "brown", cextext = 1.2,
           main = paste("plot symbols :  points (...  pch = *, cex =",
                        cex,")"))
  {
    nex <- length(extras)
    np  <- 26 + nex
    ipch <- 0:(np-1)
    k <- floor(sqrt(np))
    dd <- c(-1,1)/2
```

```
    rx <- dd + range(ix <- ipch %/% k)
    ry <- dd + range(iy <- 3 + (k-1)- ipch %% k)
    pch <- as.list(ipch) # list with integers & strings
    if(nex > 0) pch[26+ 1:nex] <- as.list(extras)
    plot(rx, ry, type = "n", axes  =  FALSE, xlab = "", ylab = "", main = main)
    abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
    for(i in 1:np) {
      pc <- pch[[i]]
      ## 'col' symbols with a 'bg'-colored interior (where available) :
      points(ix[i], iy[i], pch = pc, col = col, bg = bg, cex = cex)
      if(cextext > 0)
          text(ix[i] - 0.3, iy[i], pc, col = coltext, cex = cextext)
    }
  }

pchShow()
pchShow(c("o","O","0"), cex = 2.5)
pchShow(NULL, cex = 4, cextext = 0, main = NULL)


## ------------ test code for various pch specifications -------------
# Try this in various font families (including Hershey)
# and locales.  Use sign = -1 asserts we want Latin-1.
# Standard cases in a MBCS locale will not plot the top half.
TestChars <- function(sign = 1, font = 1, ...)
{
   MB <- l10n_info()$MBCS
   r <- if(font == 5) { sign <- 1; c(32:126, 160:254)
       } else if(MB) 32:126 else 32:255
   if (sign == -1) r <- c(32:126, 160:255)
   par(pty = "s")
   plot(c(-1,16), c(-1,16), type = "n", xlab = "", ylab = "",
        xaxs = "i", yaxs = "i",
        main = sprintf("sign = %d, font = %d", sign, font))
   grid(17, 17, lty = 1) ; mtext(paste("MBCS:", MB))
   for(i in r) try(points(i%%16, i%/%16, pch = sign*i, font = font,...))
}
TestChars()
try(TestChars(sign = -1))
TestChars(font = 5)  # Euro might be at 160 (0+10*16).
                     # macOS has apple at 240 (0+15*16).
try(TestChars(-1, font = 2))  # bold
```

---

polygon *Polygon Drawing*

Description

polygon draws the polygons whose vertices are given in x and y.

Usage

```
polygon(x, y = NULL, density = NULL, angle = 45,
        border = NULL, col = NA, lty = par("lty"),
        ..., fillOddEven = FALSE)
```

Arguments

x, y	vectors containing the coordinates of the vertices of the polygon.
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 nor filling whereas negative values and NA suppress shading (and so allow color filling).
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise).
col	the color for filling the polygon. The default, NA, is to leave polygons unfilled, unless density is specified. (For back-compatibility, NULL is equivalent to NA.) If density is specified with a positive value this gives the color of the shading lines.
border	the color to draw the border. The default, NULL, means to use par("fg"). Use border = NA to omit borders. For compatibility with S, border can also be logical, in which case FALSE is equivalent to NA (borders omitted) and TRUE is equivalent to NULL (use the foreground colour),
lty	the line type to be used, as in par.
...	graphical parameters such as xpd, lend, ljoin and lmitre can be given as arguments.
fillOddEven	logical controlling the polygon shading mode: see below for details. Default FALSE.

Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ... See xy.coords.

It is assumed that the polygon is to be closed by joining the last point to the first point.

The coordinates can contain missing values. The behaviour is similar to that of lines, except that instead of breaking a line into several lines, NA values break the polygon into several complete polygons (including closing the last point to the first point). See the examples below.

When multiple polygons are produced, the values of density, angle, col, border, and lty are recycled in the usual manner.

Shading of polygons is only implemented for linear plots: if either axis is on log scale then shading is omitted, with a warning.

Bugs

Self-intersecting polygons may be filled using either the "odd-even" or "non-zero" rule. These fill a region if the polygon border encircles it an odd or non-zero number of times, respectively. Shading lines are handled internally by R according to the fillOddEven argument, but device-based solid fills depend on the graphics device. The windows, pdf and postscript devices have their own fillOddEven argument to control this.

Author(s)

The code implementing polygon shading was donated by Kevin Buhr <buhr@stat.wisc.edu>.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

`segments` for even more flexibility, `lines`, `rect`, `box`, `abline`.

`par` for how to specify colors.

Examples

```
x <- c(1:9, 8:1)
y <- c(1, 2*(5:3), 2, -1, 17, 9, 8, 2:9)
op <- par(mfcol = c(3, 1))
for(xpd in c(FALSE, TRUE, NA)) {
  plot(1:10, main = paste("xpd =", xpd))
  box("figure", col = "pink", lwd = 3)
  polygon(x, y, xpd = xpd, col = "orange", lty = 2, lwd = 2, border = "red")
}
par(op)

n <- 100
xx <- c(0:n, n:0)
yy <- c(c(0, cumsum(stats::rnorm(n))), rev(c(0, cumsum(stats::rnorm(n)))))
plot   (xx, yy, type = "n", xlab = "Time", ylab = "Distance")
polygon(xx, yy, col = "gray", border = "red")
title("Distance Between Brownian Motions")

# Multiple polygons from NA values
# and recycling of col, border, and lty
op <- par(mfrow = c(2, 1))
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,1,2,1,2,1),
        col = c("red", "blue"),
        border = c("green", "yellow"),
        lwd = 3, lty = c("dashed", "solid"))
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,NA,2,1,2,1),
        col = c("red", "blue"),
        border = c("green", "yellow"),
        lwd = 3, lty = c("dashed", "solid"))
par(op)

# Line-shaded polygons
plot(c(1, 9), 1:2, type = "n")
polygon(1:9, c(2,1,2,1,NA,2,1,2,1),
        density = c(10, 20), angle = c(-45, 45))
```

`polypath` *Path Drawing*

Description

path draws a path whose vertices are given in x and y.

Usage

```
polypath(x, y = NULL,
         border = NULL, col = NA, lty = par("lty"),
         rule = "winding", ...)
```

Arguments

x, y	vectors containing the coordinates of the vertices of the path.
col	the color for filling the path. The default, NA, is to leave paths unfilled.
border	the color to draw the border. The default, NULL, means to use par("fg"). Use border = NA to omit borders. For compatibility with S, border can also be logical, in which case FALSE is equivalent to NA (borders omitted) and TRUE is equivalent to NULL (use the foreground colour),
lty	the line type to be used, as in par.
rule	character value specifying the path fill mode: either "winding" or "evenodd".
...	graphical parameters such as xpd, lend, ljoin and lmitre can be given as arguments.

Details

The coordinates can be passed in a plotting structure (a list with x and y components), a two-column matrix, ... See xy.coords.

It is assumed that the path is to be closed by joining the last point to the first point.

The coordinates can contain missing values. The behaviour is similar to that of polygon, except that instead of breaking a polygon into several polygons, NA values break the path into several sub-paths (including closing the last point to the first point in each sub-path). See the examples below.

The distinction between a path and a polygon is that the former can contain holes, as interpreted by the fill rule; these fill a region if the path border encircles it an odd or non-zero number of times, respectively.

Hatched shading (as implemented for polygon()) is not (currently) supported.

Not all graphics devices support this function: for example xfig and pictex do not.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

segments for even more flexibility, lines, rect, box, polygon.

par for how to specify colors.

Examples

```
plotPath <- function(x, y, col = "grey", rule = "winding") {
    plot.new()
    plot.window(range(x, na.rm = TRUE), range(y, na.rm = TRUE))
    polypath(x, y, col = col, rule = rule)
    if (!is.na(col))
        mtext(paste("Rule:", rule), side = 1, line = 0)
}

plotRules <- function(x, y, title) {
    plotPath(x, y)
    plotPath(x, y, rule = "evenodd")
    mtext(title, side = 3, line = 0)
    plotPath(x, y, col = NA)
}

op <- par(mfrow = c(5, 3), mar = c(2, 1, 1, 1))

plotRules(c(.1, .1, .9, .9, NA, .2, .2, .8, .8),
          c(.1, .9, .9, .1, NA, .2, .8, .8, .2),
          "Nested rectangles, both clockwise")
plotRules(c(.1, .1, .9, .9, NA, .2, .8, .8, .2),
          c(.1, .9, .9, .1, NA, .2, .2, .8, .8),
          "Nested rectangles, outer clockwise, inner anti-clockwise")
plotRules(c(.1, .1, .4, .4, NA, .6, .9, .9, .6),
          c(.1, .4, .4, .1, NA, .6, .6, .9, .9),
          "Disjoint rectangles")
plotRules(c(.1, .1, .6, .6, NA, .4, .4, .9, .9),
          c(.1, .6, .6, .1, NA, .4, .9, .9, .4),
          "Overlapping rectangles, both clockwise")
plotRules(c(.1, .1, .6, .6, NA, .4, .9, .9, .4),
          c(.1, .6, .6, .1, NA, .4, .4, .9, .9),
          "Overlapping rectangles, one clockwise, other anti-clockwise")

par(op)
```

---

rasterImage *Draw One or More Raster Images*

Description

`rasterImage` draws a raster image at the given locations and sizes.

Usage

```
rasterImage(image,
            xleft, ybottom, xright, ytop,
            angle = 0, interpolate = TRUE, ...)
```

Arguments

`image`	a `raster` object, or an object that can be coerced to one by `as.raster`.
`xleft`	a vector (or scalar) of left x positions.

ybottom	a vector (or scalar) of bottom y positions.
xright	a vector (or scalar) of right x positions.
ytop	a vector (or scalar) of top y positions.
angle	angle of rotation (in degrees, anti-clockwise from positive x-axis, about the bottom-left corner).
interpolate	a logical vector (or scalar) indicating whether to apply linear interpolation to the image when drawing.
...	graphical parameters.

Details

The positions supplied, i.e., `xleft, ...`, are relative to the current plotting region. If the x-axis goes from 100 to 200 then `xleft` should be larger than 100 and `xright` should be less than 200. The position vectors will be recycled to the length of the longest.

Plotting raster images is not supported on all devices and may have limitations where supported, for example (e.g., for `postscript` and `X11(type = "Xlib")` is restricted to opaque colors). Problems with the rendering of raster images have been reported by users of `windows()` devices under Remote Desktop, at least under its default settings.

You should not expect a raster image to be re-sized when an on-screen device is re-sized: whether it is is device-dependent.

See Also

`rect`, `polygon`, and `segments` and others for flexible ways to draw shapes.

`dev.capabilities` to see if it is supported.

Examples

```
require(grDevices)
## set up the plot region:
op <- par(bg = "thistle")
plot(c(100, 250), c(300, 450), type = "n", xlab = "", ylab = "")
image <- as.raster(matrix(0:1, ncol = 5, nrow = 3))
rasterImage(image, 100, 300, 150, 350, interpolate = FALSE)
rasterImage(image, 100, 400, 150, 450)
rasterImage(image, 200, 300, 200 + xinch(.5), 300 + yinch(.3),
            interpolate = FALSE)
rasterImage(image, 200, 400, 250, 450, angle = 15, interpolate = FALSE)
par(op)
```

---

rect *Draw One or More Rectangles*

Description

`rect` draws a rectangle (or sequence of rectangles) with the given coordinates, fill and border colors.

Usage

```
rect(xleft, ybottom, xright, ytop, density = NULL, angle = 45,
     col = NA, border = NULL, lty = par("lty"), lwd = par("lwd"),
     ...)
```

Arguments

xleft	a vector (or scalar) of left x positions.
ybottom	a vector (or scalar) of bottom y positions.
xright	a vector (or scalar) of right x positions.
ytop	a vector (or scalar) of top y positions.
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.
col	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 means par("fg"). Use border = NA to omit borders. 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.
...	graphical parameters such as xpd, lend, ljoin and lmitre can be given as arguments.

Details

The positions supplied, i.e., xleft, ..., are relative to the current plotting region. If the x-axis goes from 100 to 200 then xleft must be larger than 100 and xright must be less than 200. The position vectors will be recycled to the length of the longest.

It is a graphics primitive used in hist, barplot, legend, etc.

See Also

box for the standard box around the plot; polygon and segments for flexible line drawing.

par for how to specify colors.

Examples

```
require(grDevices)
## set up the plot region:
op <- par(bg = "thistle")
plot(c(100, 250), c(300, 450), type = "n", xlab = "", ylab = "",
     main = "2 x 11 rectangles; 'rect(100+i,300+i,  150+i,380+i)'")
i <- 4*(0:10)
## draw rectangles with bottom left (100, 300)+i
## and top right (150, 380)+i
rect(100+i, 300+i, 150+i, 380+i, col = rainbow(11, start = 0.7, end = 0.1))
rect(240-i, 320+i, 250-i, 410+i, col = heat.colors(11), lwd = i/5)
```

```
## Background alternating  ( transparent / "bg" ) :
j <- 10*(0:5)
rect(125+j, 360+j,   141+j, 405+j/2, col = c(NA,0),
     border = "gold", lwd = 2)
rect(125+j, 296+j/2, 141+j, 331+j/5, col = c(NA,"midnightblue"))
mtext("+  2 x 6 rect(*, col = c(NA,0)) and  col = c(NA,\"m..blue\")")

## an example showing colouring and shading
plot(c(100, 200), c(300, 450), type= "n", xlab = "", ylab = "")
rect(100, 300, 125, 350) # transparent
rect(100, 400, 125, 450, col = "green", border = "blue") # coloured
rect(115, 375, 150, 425, col = par("bg"), border = "transparent")
rect(150, 300, 175, 350, density = 10, border = "red")
rect(150, 400, 175, 450, density = 30, col = "blue",
     angle = -30, border = "transparent")

legend(180, 450, legend = 1:4, fill = c(NA, "green", par("fg"), "blue"),
       density = c(NA, NA, 10, 30), angle = c(NA, NA, 30, -30))

par(op)
```

`rug` *Add a Rug to a Plot*

Description

Adds a *rug* representation (1-d plot) of the data to the plot.

Usage

```
rug(x, ticksize = 0.03, side = 1, lwd = 0.5, col = par("fg"),
    quiet = getOption("warn") < 0, ...)
```

Arguments

`x`	A numeric vector
`ticksize`	The length of the ticks making up the 'rug'. Positive lengths give inwards ticks.
`side`	On which side of the plot box the rug will be plotted. Normally 1 (bottom) or 3 (top).
`lwd`	The line width of the ticks. Some devices will round the default width up to `1`.
`col`	The colour the ticks are plotted in.
`quiet`	logical indicating if there should be a warning about clipped values.
`...`	further arguments, passed to `axis`, such as `line` or `pos` for specifying the location of the rug.

Details

Because of the way `rug` is implemented, only values of `x` that fall within the plot region are included. There will be a warning if any finite values are omitted, but non-finite values are omitted silently.

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S.* Wadsworth & Brooks/Cole.

See Also

`jitter` which you may want for ties in `x`.

Examples

```
require(stats)  # both 'density' and its default method
with(faithful, {
    plot(density(eruptions, bw = 0.15))
    rug(eruptions)
    rug(jitter(eruptions, amount = 0.01), side = 3, col = "light blue")
})
```

---

screen *Creating and Controlling Multiple Screens on a Single Device*

Description

`split.screen` defines a number of regions within the current device which can, to some extent, be treated as separate graphics devices. It is useful for generating multiple plots on a single device. Screens can themselves be split, allowing for quite complex arrangements of plots.

`screen` is used to select which screen to draw in.

`erase.screen` is used to clear a single screen, which it does by filling with the background colour.

`close.screen` removes the specified screen definition(s).

Usage

```
split.screen(figs, screen, erase = TRUE)
screen(n = , new = TRUE)
erase.screen(n = )
close.screen(n, all.screens = FALSE)
```

Arguments

`figs`	a two-element vector describing the number of rows and the number of columns in a screen matrix *or* a matrix with 4 columns. If a matrix, then each row describes a screen with values for the left, right, bottom, and top of the screen (in that order) in NDC units, that is 0 at the lower left corner of the device surface, and 1 at the upper right corner.
`screen`	a number giving the screen to be split. It defaults to the current screen if there is one, otherwise the whole device region.
`erase`	logical: should the selected screen be cleared?
`n`	a number indicating which screen to prepare for drawing (`screen`), erase (`erase.screen`), or close (`close.screen`). (`close.screen` will accept a vector of screen numbers.)
`new`	logical value indicating whether the screen should be erased as part of the preparation for drawing in the screen.
`all.screens`	logical value indicating whether all of the screens should be closed.

Details

The first call to `split.screen` places R into split-screen mode. The other split-screen functions only work within this mode. While in this mode, certain other commands should be avoided (see the Warnings section below). Split-screen mode is exited by the command `close.screen(all = TRUE)`.

If the current screen is closed, `close.screen` sets the current screen to be the next larger screen number if there is one, otherwise to the first available screen.

Value

`split.screen(*)` returns a vector of screen numbers for the newly-created screens. With no arguments, `split.screen()` returns a vector of valid screen numbers.

`screen(n)` invisibly returns `n`, the number of the selected screen. With no arguments, `screen()` returns the number of the current screen.

`close.screen()` returns a vector of valid screen numbers.

`screen`, `erase.screen`, and `close.screen` all return `FALSE` if R is not in split-screen mode.

Warnings

The recommended way to use these functions is to completely draw a plot and all additions (i.e., points and lines) to the base plot, prior to selecting and plotting on another screen. The behavior associated with returning to a screen to add to an existing plot is unpredictable and may result in problems that are not readily visible.

These functions are totally incompatible with the other mechanisms for arranging plots on a device: `par(mfrow)`, `par(mfcol)` and `layout()`.

The functions are also incompatible with some plotting functions, such as `coplot`, which make use of these other mechanisms.

`erase.screen` will appear not to work if the background colour is transparent (as it is by default on most devices).

References

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

`par`, `layout`, `Devices`, `dev.*`

Examples

```
if (interactive()) {
par(bg = "white")           # default is likely to be transparent
split.screen(c(2, 1))       # split display into two screens
split.screen(c(1, 3), screen = 2) # now split the bottom half into 3
screen(1) # prepare screen 1 for output
plot(10:1)
screen(4) # prepare screen 4 for output
plot(10:1)
close.screen(all = TRUE)    # exit split-screen mode
```

```
split.screen(c(2, 1))       # split display into two screens
split.screen(c(1, 2), 2)    # split bottom half in two
plot(1:10)                  # screen 3 is active, draw plot
erase.screen()              # forgot label, erase and redraw
plot(1:10, ylab = "ylab 3")
screen(1)                   # prepare screen 1 for output
plot(1:10)
screen(4)                   # prepare screen 4 for output
plot(1:10, ylab = "ylab 4")
screen(1, FALSE)            # return to screen 1, but do not clear
plot(10:1, axes = FALSE, lty = 2, ylab = "")  # overlay second plot
axis(4)                     # add tic marks to right-hand axis
title("Plot 1")
close.screen(all = TRUE)    # exit split-screen mode
}
```

`segments` *Add Line Segments to a Plot*

Description

Draw line segments between pairs of points.

Usage

```
segments(x0, y0, x1 = x0, y1 = y0,
         col = par("fg"), lty = par("lty"), lwd = par("lwd"),
         ...)
```

Arguments

`x0`, `y0`	coordinates of points **from** which to draw.
`x1`, `y1`	coordinates of points **to** which to draw. At least one must be supplied.
`col`, `lty`, `lwd`	graphical parameters as in `par`, possibly vectors. `NA` values in `col` cause the segment to be omitted.
`...`	further graphical parameters (from `par`), such as `xpd` and the line characteristics `lend`, `ljoin` and `lmitre`.

Details

For each `i`, a line segment is drawn between the point `(x0[i], y0[i])` and the point `(x1[i], y1[i])`. The coordinate vectors will be recycled to the length of the longest.

The graphical parameters `col`, `lty` and `lwd` can be vectors of length greater than one and will be recycled if necessary.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

arrows, polygon for slightly easier and less flexible line drawing, and lines for the usual polygons.

Examples

```
x <- stats::runif(12); y <- stats::rnorm(12)
i <- order(x, y); x <- x[i]; y <- y[i]
plot(x, y, main = "arrows(.) and segments(.)")
## draw arrows from point to point :
s <- seq(length(x)-1)  # one shorter than data
arrows(x[s], y[s], x[s+1], y[s+1], col= 1:3)
s <- s[-length(s)]
segments(x[s], y[s], x[s+2], y[s+2], col= 'pink')
```

---

smoothScatter *Scatterplots with Smoothed Densities Color Representation*

Description

smoothScatter produces a smoothed color density representation of a scatterplot, obtained through a (2D) kernel density estimate.

Usage

```
smoothScatter(x, y = NULL, nbin = 128, bandwidth,
              colramp = colorRampPalette(c("white", blues9)),
              nrpoints = 100, ret.selection = FALSE,
              pch = ".", cex = 1, col = "black",
              transformation = function(x) x^.25,
              postPlotHook = box,
              xlab = NULL, ylab = NULL, xlim, ylim,
              xaxs = par("xaxs"), yaxs = par("yaxs"), ...)
```

Arguments

x, y	the x and y arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function xy.coords for details. If supplied separately, they must be of the same length.
nbin	numeric vector of length one (for both directions) or two (for x and y separately) specifying the number of equally spaced grid points for the density estimation; directly used as gridsize in bkde2D().
bandwidth	numeric vector (length 1 or 2) of smoothing bandwidth(s). If missing, a more or less useful default is used. bandwidth is subsequently passed to function bkde2D.
colramp	function accepting an integer n as an argument and returning n colors.
nrpoints	number of points to be superimposed on the density image. The first nrpoints points from those areas of lowest regional densities will be plotted. Adding points to the plot allows for the identification of outliers. If all points are to be plotted, choose nrpoints = Inf.

ret.selection	logical indicating to return the ordered indices of "low density" points if nrpoints > 0.
pch, cex, col	arguments passed to points, when nrpoints > 0: point symbol, character expansion factor and color, see also par.
transformation	function mapping the density scale to the color scale.
postPlotHook	either NULL or a function which will be called (with no arguments) after image.
xlab, ylab	character strings to be used as axis labels, passed to image.
xlim, ylim	numeric vectors of length 2 specifying axis limits.
xaxs, yaxs, ...	further arguments passed to image, e.g., add=TRUE or useRaster=TRUE.

Details

smoothScatter produces a smoothed version of a scatter plot. Two dimensional (kernel density) smoothing is performed by bkde2D from package **KernSmooth**. See the examples for how to use this function together with pairs.

Value

If ret.selection is true, a vector of integers of length nrpoints (or smaller, if there are less finite points inside xlim and ylim) with the indices of the low-density points drawn, ordered with lowest density first.

Author(s)

Florian Hahne at FHCRC, originally

See Also

bkde2D from package **KernSmooth**; densCols which uses the same smoothing computations and blues9 in package **grDevices**.

scatter.smooth adds a loess regression smoother to a scatter plot.

Examples

```
## A largish data set
n <- 10000
x1  <- matrix(rnorm(n), ncol = 2)
x2  <- matrix(rnorm(n, mean = 3, sd = 1.5), ncol = 2)
x   <- rbind(x1, x2)

oldpar <- par(mfrow = c(2, 2), mar=.1+c(3,3,1,1), mgp = c(1.5, 0.5, 0))
smoothScatter(x, nrpoints = 0)
smoothScatter(x)

## a different color scheme:
Lab.palette <- colorRampPalette(c("blue", "orange", "red"), space = "Lab")
i.s <- smoothScatter(x, colramp = Lab.palette,
                     ## pch=NA: do not draw them
                     nrpoints = 250, ret.selection=TRUE)
## label the 20 very lowest-density points,the "outliers" (with obs.number):
i.20 <- i.s[1:20]
text(x[i.20,], labels = i.20, cex= 0.75)
```

```
## somewhat similar, using identical smoothing computations,
## but considerably *less* efficient for really large data:
plot(x, col = densCols(x), pch = 20)

## use with pairs:
par(mfrow = c(1, 1))
y <- matrix(rnorm(40000), ncol = 4) + 3*rnorm(10000)
y[, c(2,4)] <-  -y[, c(2,4)]
pairs(y, panel = function(...) smoothScatter(..., nrpoints = 0, add = TRUE),
      gap = 0.2)

par(oldpar)
```

---

spineplot *Spine Plots and Spinograms*

Description

Spine plots are a special cases of mosaic plots, and can be seen as a generalization of stacked (or highlighted) bar plots. Analogously, spinograms are an extension of histograms.

Usage

```
spineplot(x, ...)

## Default S3 method:
spineplot(x, y = NULL,
          breaks = NULL, tol.ylab = 0.05, off = NULL,
          ylevels = NULL, col = NULL,
          main = "", xlab = NULL, ylab = NULL,
          xaxlabels = NULL, yaxlabels = NULL,
          xlim = NULL, ylim = c(0, 1), axes = TRUE, weights = NULL, ...)

## S3 method for class 'formula'
spineplot(formula, data = NULL,
          breaks = NULL, tol.ylab = 0.05, off = NULL,
          ylevels = NULL, col = NULL,
          main = "", xlab = NULL, ylab = NULL,
          xaxlabels = NULL, yaxlabels = NULL,
          xlim = NULL, ylim = c(0, 1), axes = TRUE, ...,
          subset = NULL, weights = NULL, drop.unused.levels = FALSE)
```

Arguments

x	an object, the default method expects either a single variable (interpreted to be the explanatory variable) or a 2-way table. See details.
y	a "factor" interpreted to be the dependent variable
formula	a "formula" of type y ~ x with a single dependent "factor" and a single explanatory variable.
data	an optional data frame.

breaks	if the explanatory variable is numeric, this controls how it is discretized. breaks is passed to hist and can be a list of arguments.
tol.ylab	convenience tolerance parameter for y-axis annotation. If the distance between two labels drops under this threshold, they are plotted equidistantly.
off	vertical offset between the bars (in per cent). It is fixed to 0 for spinograms and defaults to 2 for spine plots.
ylevels	a character or numeric vector specifying in which order the levels of the dependent variable should be plotted.
col	a vector of fill colors of the same length as levels(y). The default is to call gray.colors.
main, xlab, ylab	character strings for annotation
xaxlabels, yaxlabels	character vectors for annotation of x and y axis. Default to levels(y) and levels(x), respectively for the spine plot. For xaxlabels in the spinogram, the breaks are used.
xlim, ylim	the range of x and y values with sensible defaults.
axes	logical. If FALSE all axes (including those giving level names) are suppressed.
weights	numeric. A vector of frequency weights for each observation in the data. If NULL all weights are implicitly assumed to be 1. If x is already a 2-way table, the weights are ignored.
...	additional arguments passed to rect.
subset	an optional vector specifying a subset of observations to be used for plotting.
drop.unused.levels	should factors have unused levels dropped? Defaults to FALSE.

Details

spineplot creates either a spinogram or a spine plot. It can be called via spineplot(x, y) or spineplot(y ~ x) where y is interpreted to be the dependent variable (and has to be categorical) and x the explanatory variable. x can be either categorical (then a spine plot is created) or numerical (then a spinogram is plotted). Additionally, spineplot can also be called with only a single argument which then has to be a 2-way table, interpreted to correspond to table(x, y).

Both, spine plots and spinograms, are essentially mosaic plots with special formatting of spacing and shading. Conceptually, they plot $P(y|x)$ against $P(x)$. For the spine plot (where both x and y are categorical), both quantities are approximated by the corresponding empirical relative frequencies. For the spinogram (where x is numerical), x is first discretized (by calling hist with breaks argument) and then empirical relative frequencies are taken.

Thus, spine plots can also be seen as a generalization of stacked bar plots where not the heights but the widths of the bars corresponds to the relative frequencies of x. The heights of the bars then correspond to the conditional relative frequencies of y in every x group. Analogously, spinograms extend stacked histograms.

Value

The table visualized is returned invisibly.

Author(s)

Achim Zeileis <Achim.Zeileis@R-project.org>

References

Friendly, M. (1994). Mosaic displays for multi-way contingency tables. *Journal of the American Statistical Association*, **89**, 190–200. doi:10.2307/2291215.

Hartigan, J.A., and Kleiner, B. (1984). A mosaic of television ratings. *The American Statistician*, **38**, 32–35. doi:10.2307/2683556.

Hofmann, H., Theus, M. (2005), *Interactive graphics for visualizing conditional distributions*. Unpublished Manuscript.

Hummel, J. (1996). Linked bar charts: Analysing categorical data graphically. *Computational Statistics*, **11**, 23–33.

See Also

mosaicplot, hist, cdplot

Examples

```
## treatment and improvement of patients with rheumatoid arthritis
treatment <- factor(rep(c(1, 2), c(43, 41)), levels = c(1, 2),
                    labels = c("placebo", "treated"))
improved <- factor(rep(c(1, 2, 3, 1, 2, 3), c(29, 7, 7, 13, 7, 21)),
                   levels = c(1, 2, 3),
                   labels = c("none", "some", "marked"))

## (dependence on a categorical variable)
(spineplot(improved ~ treatment))

## applications and admissions by department at UC Berkeley
## (two-way tables)
(spineplot(marginSums(UCBAdmissions, c(3, 2)),
           main = "Applications at UCB"))
(spineplot(marginSums(UCBAdmissions, c(3, 1)),
           main = "Admissions at UCB"))

## NASA space shuttle o-ring failures
fail <- factor(c(2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1,
                 1, 1, 1, 2, 1, 1, 1, 1, 1),
               levels = c(1, 2), labels = c("no", "yes"))
temperature <- c(53, 57, 58, 63, 66, 67, 67, 67, 68, 69, 70, 70,
                 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 81)

## (dependence on a numerical variable)
(spineplot(fail ~ temperature))
(spineplot(fail ~ temperature, breaks = 3))
(spineplot(fail ~ temperature, breaks = quantile(temperature)))

## highlighting for failures
spineplot(fail ~ temperature, ylevels = 2:1)
```

---

stars *Star (Spider/Radar) Plots and Segment Diagrams*

Description

Draw star plots or segment diagrams of a multivariate data set. With one single location, also draws 'spider' (or 'radar') plots.

Usage

```
stars(x, full = TRUE, scale = TRUE, radius = TRUE,
      labels = dimnames(x)[[1]], locations = NULL,
      nrow = NULL, ncol = NULL, len = 1,
      key.loc = NULL, key.labels = dimnames(x)[[2]],
      key.xpd = TRUE,
      xlim = NULL, ylim = NULL, flip.labels = NULL,
      draw.segments = FALSE,
      col.segments = 1:n.seg, col.stars = NA, col.lines = NA,
      axes = FALSE, frame.plot = axes,
      main = NULL, sub = NULL, xlab = "", ylab = "",
      cex = 0.8, lwd = 0.25, lty = par("lty"), xpd = FALSE,
      mar = pmin(par("mar"),
                 1.1+ c(2*axes+ (xlab != ""),
                 2*axes+ (ylab != ""), 1, 0)),
      add = FALSE, plot = TRUE, ...)
```

Arguments

x	matrix or data frame of data. One star or segment plot will be produced for each row of x. Missing values (NA) are allowed, but they are treated as if they were 0 (after scaling, if relevant).
full	logical flag: if TRUE, the segment plots will occupy a full circle. Otherwise, they occupy the (upper) semicircle only.
scale	logical flag: if TRUE, the columns of the data matrix are scaled independently so that the maximum value in each column is 1 and the minimum is 0. If FALSE, the presumption is that the data have been scaled by some other algorithm to the range $[0, 1]$.
radius	logical flag: in TRUE, the radii corresponding to each variable in the data will be drawn.
labels	vector of character strings for labeling the plots. Unlike the S function stars, no attempt is made to construct labels if labels = NULL.
locations	Either two column matrix with the x and y coordinates used to place each of the segment plots; or numeric of length 2 when all plots should be superimposed (for a 'spider plot'). By default, locations = NULL, the segment plots will be placed in a rectangular grid.
nrow, ncol	integers giving the number of rows and columns to use when locations is NULL. By default, nrow == ncol, a square layout will be used.
len	scale factor for the length of radii or segments.
key.loc	vector with x and y coordinates of the unit key.
key.labels	vector of character strings for labeling the segments of the unit key. If omitted, the second component of dimnames(x) is used, if available.
key.xpd	clipping switch for the unit key (drawing and labeling), see par("xpd").
xlim	vector with the range of x coordinates to plot.

ylim	vector with the range of y coordinates to plot.
flip.labels	logical indicating if the label locations should flip up and down from diagram to diagram. Defaults to a somewhat smart heuristic.
draw.segments	logical. If TRUE draw a segment diagram.
col.segments	color vector (integer or character, see par), each specifying a color for one of the segments (variables). Ignored if draw.segments = FALSE.
col.stars	color vector (integer or character, see par), each specifying a color for one of the stars (cases). Ignored if draw.segments = TRUE.
col.lines	color vector (integer or character, see par), each specifying a color for one of the lines (cases). Ignored if draw.segments = TRUE.
axes	logical flag: if TRUE axes are added to the plot.
frame.plot	logical flag: if TRUE, the plot region is framed.
main	a main title for the plot.
sub	a subtitle for the plot.
xlab	a label for the x axis.
ylab	a label for the y axis.
cex	character expansion factor for the labels.
lwd	line width used for drawing.
lty	line type used for drawing.
xpd	logical or NA indicating if clipping should be done, see par(xpd = .).
mar	argument to par(mar = *), typically choosing smaller margins than by default.
...	further arguments, passed to the first call of plot(), see plot.default and to box() if frame.plot is true.
add	logical, if TRUE *add* stars to current plot.
plot	logical, if FALSE, nothing is plotted.

Details

Missing values are treated as 0.

Each star plot or segment diagram represents one row of the input x. Variables (columns) start on the right and wind counterclockwise around the circle. The size of the (scaled) column is shown by the distance from the center to the point on the star or the radius of the segment representing the variable.

Only one page of output is produced.

Value

Returns the locations of the plots in a two column matrix, invisibly when plot = TRUE.

Note

This code started life as spatial star plots by David A. Andrews.

Prior to R 1.4.1, scaling only shifted the maximum to 1, although documented as here.

Author(s)

Thomas S. Dye

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`symbols` for another way to draw stars and other symbols.

Examples

```
require(grDevices)
stars(mtcars[, 1:7], key.loc = c(14, 2),
      main = "Motor Trend Cars : stars(*, full = F)", full = FALSE)
stars(mtcars[, 1:7], key.loc = c(14, 1.5),
      main = "Motor Trend Cars : full stars()", flip.labels = FALSE)

## 'Spider' or 'Radar' plot:
stars(mtcars[, 1:7], locations = c(0, 0), radius = FALSE,
      key.loc = c(0, 0), main = "Motor Trend Cars", lty = 2)

## Segment Diagrams:
palette(rainbow(12, s = 0.6, v = 0.75))
stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 1.5),
      main = "Motor Trend Cars", draw.segments = TRUE)
stars(mtcars[, 1:7], len = 0.6, key.loc = c(1.5, 0),
      main = "Motor Trend Cars", draw.segments = TRUE,
      frame.plot = TRUE, nrow = 4, cex = .7)

## scale linearly (not affinely) to [0, 1]
USJudge <- apply(USJudgeRatings, 2, function(x) x/max(x))
Jnam <- row.names(USJudgeRatings)
Snam <- abbreviate(substring(Jnam, 1, regexpr("[,.]",Jnam) - 1), 7)
stars(USJudge, labels = Jnam, scale = FALSE,
      key.loc = c(13, 1.5), main = "Judge not ...", len = 0.8)
stars(USJudge, labels = Snam, scale = FALSE,
      key.loc = c(13, 1.5), radius = FALSE)

loc <- stars(USJudge, labels = NULL, scale = FALSE,
             radius = FALSE, frame.plot = TRUE,
             key.loc = c(13, 1.5), main = "Judge not ...", len = 1.2)
text(loc, Snam, col = "blue", cex = 0.8, xpd = TRUE)

## 'Segments':
stars(USJudge, draw.segments = TRUE, scale = FALSE, key.loc = c(13,1.5))

## 'Spider':
stars(USJudgeRatings, locations = c(0, 0), scale = FALSE, radius  =  FALSE,
      col.stars = 1:10, key.loc = c(0, 0), main = "US Judges rated")
## Same as above, but with colored lines instead of filled polygons.
stars(USJudgeRatings, locations = c(0, 0), scale = FALSE, radius  =  FALSE,
      col.lines = 1:10, key.loc = c(0, 0), main = "US Judges rated")
## 'Radar-Segments'
stars(USJudgeRatings[1:10,], locations = 0:1, scale = FALSE,
      draw.segments = TRUE, col.segments = 0, col.stars = 1:10, key.loc =  0:1,
      main = "US Judges 1-10 ")
palette("default")
```

```
stars(cbind(1:16, 10*(16:1)), draw.segments = TRUE,
      main = "A Joke -- do *not* use symbols on 2D data!")
```

`stem` *Stem-and-Leaf Plots*

Description

`stem` produces a stem-and-leaf plot of the values in `x`. The parameter `scale` can be used to expand the scale of the plot. A value of `scale = 2` will cause the plot to be roughly twice as long as the default.

Usage

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

Arguments

`x`	a numeric vector.
`scale`	This controls the plot length.
`width`	The desired width of plot.
`atom`	a tolerance.

Details

Infinite and missing values in `x` are discarded.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```
stem(islands)
stem(log10(islands))
```

`stripchart` *1-D Scatter Plots*

Description

`stripchart` produces one dimensional scatter plots (or dot plots) of the given data. These plots are a good alternative to `boxplot`s when sample sizes are small.

Usage

```
stripchart(x, ...)

## S3 method for class 'formula'
stripchart(x, data = NULL, dlab = NULL, ...,
           subset, na.action = NULL)


## Default S3 method:
stripchart(x, method = "overplot", jitter = 0.1, offset = 1/3,
           vertical = FALSE, group.names, add = FALSE,
           at = NULL, xlim = NULL, ylim = NULL,
           ylab = NULL, xlab = NULL, dlab = "", glab = "",
           log = "", pch = 0, col = par("fg"), cex = par("cex"),
           axes = TRUE, frame.plot = axes, ...)
```

Arguments

x	the data from which the plots are to be produced. In the default method the data can be specified as a single numeric vector, or as list of numeric vectors, each corresponding to a component plot. In the formula method, a symbolic specification of the form y ~ g can be given, indicating the observations in the vector y are to be grouped according to the levels of the factor g. NAs are allowed in the data.
data	a data.frame (or list) from which the variables in x should be taken.
subset	an optional vector specifying a subset of observations to be used for plotting.
na.action	a function which indicates what should happen when the data contain NAs. The default is to ignore missing values in either the response or the group.
...	additional parameters passed to the default method, or by it to plot.window, points, axis and title to control the appearance of the plot.
method	the method to be used to separate coincident points. The default method "overplot" causes such points to be overplotted, but it is also possible to specify "jitter" to jitter the points, or "stack" have coincident points stacked. The last method only makes sense for very granular data.
jitter	when method = "jitter" is used, jitter gives the amount of jittering applied.
offset	when stacking is used, points are stacked this many line-heights (symbol widths) apart.
vertical	when vertical is TRUE the plots are drawn vertically rather than the default horizontal.
group.names	group labels which will be printed alongside (or underneath) each plot.
add	logical, if true *add* the chart to the current plot.
at	numeric vector giving the locations where the charts should be drawn, particularly when add = TRUE; defaults to 1:n where n is the number of boxes.
ylab, xlab	labels: see title.
dlab, glab	alternate way to specify axis labels: see 'Details'.
xlim, ylim	plot limits: see plot.window.
log	on which axes to use a log scale: see plot.default

pch, col, cex Graphical parameters: see par.

axes, frame.plot
Axis control: see plot.default.

Details

Extensive examples of the use of this kind of plot can be found in Box, Hunter and Hunter (2005) or Wild and Seber (2000).

The dlab and glab labels may be used instead of xlab and ylab if those are not specified. dlab applies to the continuous data axis (the X axis unless vertical is TRUE), glab to the group axis.

References

Box G., Hunter, J. S. and Hunter, W. C. (2005). *Statistics for Experimenters: Design, Innovation, and Discovery*, second edition. New York: Wiley. ISBN: 978-0-471-71813-0.

Wild, C. and Seber, G. (2000). *Chance Encounters: A First Course in Data Analysis and Inference*. John Wiley and Sons. ISBN 0-471-32936-3.

Examples

```
x <- stats::rnorm(50)
xr <- round(x, 1)
stripchart(x) ; m <- mean(par("usr")[1:2])
text(m, 1.04, "stripchart(x, \"overplot\")")
stripchart(xr, method = "stack", add = TRUE, at = 1.2)
text(m, 1.35, "stripchart(round(x,1), \"stack\")")
stripchart(xr, method = "jitter", add = TRUE, at = 0.7)
text(m, 0.85, "stripchart(round(x,1), \"jitter\")")

stripchart(decrease ~ treatment,
    main = "stripchart(OrchardSprays)",
    vertical = TRUE, log = "y", data = OrchardSprays)

stripchart(decrease ~ treatment, at = c(1:8)^2,
    main = "stripchart(OrchardSprays)",
    vertical = TRUE, log = "y", data = OrchardSprays)
```

---

strwidth *Plotting Dimensions of Character Strings and Math Expressions*

Description

These functions compute the width or height, respectively, of the given strings or mathematical expressions s[i] on the current plotting device in *user* coordinates, *inches* or as fraction of the figure width par("fin").

Usage

```
strwidth(s, units = "user", cex = NULL, font = NULL, vfont = NULL, ...)
strheight(s, units = "user", cex = NULL, font = NULL, vfont = NULL, ...)
```

Arguments

s	a character or expression vector whose dimensions are to be determined. Other objects are coerced by as.graphicsAnnot.
units	character indicating in which units s is measured; should be one of "user", "inches", "figure"; partial matching is performed.
cex	numeric **c**haracter **ex**pansion factor; multiplied by par("cex") yields the final character size; the default NULL is equivalent to 1.
font, vfont, ...	additional information about the font, possibly including the graphics parameter "family": see text.

Details

Note that the 'height' of a string is determined only by the number of linefeeds ("\n", aka "newline"s) it contains: it is the (number of linefeeds - 1) times the line spacing plus the height of "M" in the selected font. For an expression it is the height of the bounding box as computed by plotmath. Thus in both cases it is an estimate of how far **above** the final baseline the typeset object extends. (It may also extend below the baseline.) The inter-line spacing is controlled by cex, par("lheight") and the 'point size' (but not the actual font in use).

Measurements in "user" units (the default) are only available after plot.new has been called – otherwise an error is thrown.

Value

Numeric vector with the same length as s, giving the estimate of width or height for each s[i]. NA strings are given width and height 0 (as they are not plotted).

See Also

text, nchar

Examples

```
str.ex <- c("W","w","I",".","WwI.")
op <- par(pty = "s"); plot(1:100, 1:100, type = "n")
sw <- strwidth(str.ex); sw
all.equal(sum(sw[1:4]), sw[5])
#- since the last string contains the others

sw.i <- strwidth(str.ex, "inches"); 25.4 * sw.i  # width in [mm]
unique(sw / sw.i)
# constant factor: 1 value
mean(sw.i / strwidth(str.ex, "fig")) / par('fin')[1]  # = 1: are the same

## See how letters fall in classes
##  -- depending on graphics device and font!
all.lett <- c(letters, LETTERS)
shL <- strheight(all.lett, units = "inches") * 72  # 'big points'
table(shL)  # all have same heights ...
mean(shL)/par("cin")[2] # around 0.6

(swL <- strwidth(all.lett, units = "inches") * 72)  # 'big points'
split(all.lett, factor(round(swL, 2)))
```

```
sumex <- expression(sum(x[i], i=1,n), e^{i * pi} == -1)
strwidth(sumex)
strheight(sumex)

par(op)  #- reset to previous setting
```

---

sunflowerplot *Produce a Sunflower Scatter Plot*

Description

Multiple points are plotted as 'sunflowers' with multiple leaves ('petals') such that overplotting is visualized instead of accidental and invisible.

Usage

```
sunflowerplot(x, ...)

## Default S3 method:
sunflowerplot(x, y = NULL, number, log = "", digits = 6,
              xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
              add = FALSE, rotate = FALSE,
              pch = 16, cex = 0.8, cex.fact = 1.5,
              col = par("col"), bg = NA, size = 1/8, seg.col = 2,
              seg.lwd = 1.5, ...)

## S3 method for class 'formula'
sunflowerplot(formula, data = NULL, xlab = NULL, ylab = NULL, ...,
             subset, na.action = NULL)
```

Arguments

x	numeric vector of x-coordinates of length n, say, or another valid plotting structure, as for plot.default, see also xy.coords.
y	numeric vector of y-coordinates of length n.
number	integer vector of length n. number[i] = number of replicates for (x[i], y[i]), may be 0. Default (missing(number)): compute the exact multiplicity of the points x[], y[], via xyTable().
log	character indicating log coordinate scale, see plot.default.
digits	when number is computed (i.e., not specified), x and y are rounded to digits significant digits before multiplicities are computed.
xlab, ylab	character label for x-, or y-axis, respectively.
xlim, ylim	numeric(2) limiting the extents of the x-, or y-axis.
add	logical; should the plot be added on a previous one ? Default is FALSE.
rotate	logical; if TRUE, randomly rotate the sunflowers (preventing artefacts).
pch	plotting character to be used for points (number[i]==1) and center of sunflowers.

cex	numeric; character size expansion of center points (s. pch).
cex.fact	numeric *shrinking* factor to be used for the center points *when there are flower leaves*, i.e., cex / cex.fact is used for these.
col, bg	colors for the plot symbols, passed to plot.default.
size	of sunflower leaves in inches, 1[in] := 2.54[cm]. Default: 1/8\", approximately 3.2mm.
seg.col	color to be used for the **seg**ments which make the sunflowers leaves, see par(col=); col = "gold" reminds of real sunflowers.
seg.lwd	numeric; the line width for the leaves' segments.
...	further arguments to plot [if add = FALSE], or to be passed to or from another method.
formula	a formula, such as y ~ x.
data	a data.frame (or list) from which the variables in formula should be taken.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is to ignore case with missing values.

Details

This is a generic function with default and formula methods.

For number[i] == 1, a (slightly enlarged) usual plotting symbol (pch) is drawn. For number[i] > 1, a small plotting symbol is drawn and number[i] equi-angular 'rays' emanate from it.

If rotate = TRUE and number[i] >= 2, a random direction is chosen (instead of the y-axis) for the first ray. The goal is to jitter the orientations of the sunflowers in order to prevent artefactual visual impressions.

Value

A list with three components of same length,

x	x coordinates
y	y coordinates
number	number

Use xyTable() (from package **grDevices**) if you are only interested in this return value.

Side Effects

A scatter plot is drawn with 'sunflowers' as symbols.

Author(s)

Andreas Ruckstuhl, Werner Stahel, Martin Maechler, Tim Hesterberg, 1989–1993. Port to R by Martin Maechler <maechler@stat.math.ethz.ch>.

References

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983). *Graphical Methods for Data Analysis*. Wadsworth.

Schilling, M. F. and Watkins, A. E. (1994). A suggestion for sunflower plots. *The American Statistician*, **48**, 303–305. doi:10.2307/2684839.

Murrell, P. (2005). *R Graphics*. Chapman & Hall/CRC Press.

See Also

density, xyTable

Examples

```
require(stats) # for rnorm
require(grDevices)

## 'number' is computed automatically:
sunflowerplot(iris[, 3:4])
## Imitating Chambers et al, p.109, closely:
sunflowerplot(iris[, 3:4], cex = .2, cex.fact = 1, size = .035, seg.lwd = .8)
## or
sunflowerplot(Petal.Width ~ Petal.Length, data = iris,
              cex = .2, cex.fact = 1, size = .035, seg.lwd = .8)


sunflowerplot(x = sort(2*round(rnorm(100))), y = round(rnorm(100), 0),
              main = "Sunflower Plot of Rounded N(0,1)")
## Similarly using a "xyTable" argument:
xyT <- xyTable(x = sort(2*round(rnorm(100))), y = round(rnorm(100), 0),
               digits = 3)
utils::str(xyT, vec.len = 20)
sunflowerplot(xyT, main = "2nd Sunflower Plot of Rounded N(0,1)")

## A 'marked point process' {explicit 'number' argument}:
sunflowerplot(rnorm(100), rnorm(100), number = rpois(n = 100, lambda = 2),
              main = "Sunflower plot (marked point process)",
              rotate = TRUE, col = "blue4")
```

---

symbols *Draw Symbols (Circles, Squares, Stars, Thermometers, Boxplots)*

Description

This function draws symbols on a plot. One of six symbols; *circles*, *squares*, *rectangles*, *stars*, *thermometers*, and *boxplots*, can be plotted at a specified set of x and y coordinates. Specific aspects of the symbols, such as relative size, can be customized by additional parameters.

Usage

```
symbols(x, y = NULL, circles, squares, rectangles, stars,
        thermometers, boxplots, inches = TRUE, add = FALSE,
        fg = par("col"), bg = NA,
        xlab = NULL, ylab = NULL, main = NULL,
        xlim = NULL, ylim = NULL, ...)
```

Arguments

x, y	the x and y co-ordinates for the centres of the symbols. They can be specified in any way which is accepted by `xy.coords`.
circles	a vector giving the radii of the circles.
squares	a vector giving the length of the sides of the squares.
rectangles	a matrix with two columns. The first column gives widths and the second the heights of rectangles.
stars	a matrix with three or more columns giving the lengths of the rays from the center of the stars. NA values are replaced by zeroes.
thermometers	a matrix with three or four columns. The first two columns give the width and height of the thermometer symbols. If there are three columns, the third is taken as a proportion: the thermometers are filled (using colour fg) from their base to this proportion of their height. If there are four columns, the third and fourth columns are taken as proportions and the thermometers are filled between these two proportions of their heights. The part of the box not filled in fg will be filled in the background colour (default transparent) given by bg.
boxplots	a matrix with five columns. The first two columns give the width and height of the boxes, the next two columns give the lengths of the lower and upper whiskers and the fifth the proportion (with a warning if not in [0,1]) of the way up the box that the median line is drawn.
inches	TRUE, FALSE or a positive number. See 'Details'.
add	if add is TRUE, the symbols are added to an existing plot, otherwise a new plot is created.
fg	colour(s) the symbols are to be drawn in.
bg	if specified, the symbols are filled with colour(s), the vector bg being recycled to the number of symbols. The default is to leave the symbols unfilled.
xlab	the x label of the plot if add is not true. Defaults to the `deparse`d expression used for x.
ylab	the y label of the plot. Unused if add = TRUE.
main	a main title for the plot. Unused if add = TRUE.
xlim	numeric vector of length 2 giving the x limits for the plot. Unused if add = TRUE.
ylim	numeric vector of length 2 giving the y limits for the plot. Unused if add = TRUE.
...	graphics parameters can also be passed to this function, as can the plot aspect ratio asp (see `plot.window`).

Details

Observations which have missing coordinates or missing size parameters are not plotted. The exception to this is *stars*. In that case, the length of any ray which is NA is reset to zero.

Argument inches controls the sizes of the symbols. If TRUE (the default), the symbols are scaled so that the largest dimension of any symbol is one inch. If a positive number is given the symbols are scaled to make largest dimension this size in inches (so TRUE and 1 are equivalent). If inches is FALSE, the units are taken to be those of the appropriate axes. (For circles, squares and stars the units of the x axis are used. For boxplots, the lengths of the whiskers are regarded as dimensions alongside width and height when scaling by inches, and are otherwise interpreted in the units of the y axis.)

Circles of radius zero are plotted at radius one pixel (which is device-dependent). Circles of a very small non-zero radius may or may not be visible, and may be smaller than circles of radius zero. On `windows` devices circles are plotted at radius at least one pixel as some Windows versions omit smaller circles.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

W. S. Cleveland (1985) *The Elements of Graphing Data.* Monterey, California: Wadsworth.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

`stars` for drawing *stars* with a bit more flexibility.

If you are thinking about doing 'bubble plots' by `symbols(*, circles=*)`, you should *really* consider using `sunflowerplot` instead.

Examples

```
require(stats); require(grDevices)
x <- 1:10
y <- sort(10*runif(10))
z <- runif(10)
z3 <- cbind(z, 2*runif(10), runif(10))
symbols(x, y, thermometers = cbind(.5, 1, z), inches = .5, fg = 1:10)
symbols(x, y, thermometers = z3, inches = FALSE)
text(x, y, apply(format(round(z3, digits = 2)), 1, paste, collapse = ","),
     adj = c(-.2,0), cex = .75, col = "purple", xpd = NA)

## Note that  example(trees)  shows more sensible plots!
N <- nrow(trees)
with(trees, {
## Girth is diameter in inches
symbols(Height, Volume, circles = Girth/24, inches = FALSE,
        main = "Trees' Girth") # xlab and ylab automatically
## Colours too:
op <- palette(rainbow(N, end = 0.9))
symbols(Height, Volume, circles = Girth/16, inches = FALSE, bg = 1:N,
        fg = "gray30", main = "symbols(*, circles = Girth/16, bg = 1:N)")
palette(op)
})
```

---

text *Add Text to a Plot*

Description

`text` draws the strings given in the vector `labels` at the coordinates given by `x` and `y`. `y` may be missing since `xy.coords(x, y)` is used for construction of the coordinates.

Usage

```
text(x, ...)

## Default S3 method:
 text(x, y = NULL, labels = seq_along(x$x), adj = NULL,
     pos = NULL, offset = 0.5, vfont = NULL,
     cex = 1, col = NULL, font = NULL, ...)
```

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.
`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`	one or two values in $[0, 1]$ which specify the x (and optionally y) adjustment ('justification') of the labels, with 0 for left/bottom, 1 for right/top, and 0.5 for centered. On most devices values outside $[0, 1]$ will also work. See below.
`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 `(x,y)` coordinates.
`offset`	when `pos` is specified, this value controls the distance ('offset') of the text 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 **c**haracter **ex**pansion 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()`.
`...`	further graphical parameters (from `par`), such as `srt`, `family` and `xpd`.

Details

`labels` must be of type `character` or `expression` (or be coercible to such a type). In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, Greek letters, fractions, etc.

`adj` allows *adj*ustment of the text position with respect to `(x, y)`. Values of 0, 0.5, and 1 specify that `(x, y)` should align with the left/bottom, middle and right/top of the text, respectively. A value of `NA` means "centre", which is the same as 0.5 for horizontal justification, but includes descenders for vertical justification (where 0.5 does not). The default is for centered text, although the default horizontal justification is taken from `par(adj)`, i.e., the default is `adj = c(par("adj"), NA)`. If only one value is provided, it is applied to adjust `x` *only*, i.e., when `length(adj) == 1L`, `adj` is applied as `adj = c(adj, NA)`. Accurate vertical centering needs character metric information on individual characters which is only available on some devices. Vertical alignment is done slightly differently for character strings and for expressions: `adj = c(0,0)` means to left-justify and to align on the baseline for strings but on the bottom of the bounding box for expressions. This also affects vertical centering: for strings the centering excludes any descenders whereas for expressions it includes them.

The pos and offset arguments can be used in conjunction with values returned by identify to recreate an interactively labelled plot.

Text can be rotated by using graphical parameters srt (see par). When adj is specified, a non-zero srt rotates the label about (x, y). If pos is specified, srt rotates the text about the point on its bounding box which is closest to (x, y): top center for pos = 1, right center for pos = 2, bottom center for pos = 3, and left center for pos = 4. The pos interface is not as useful for rotated text because the result is no longer centered vertically or horizontally with respect to (x, y). At present there is no interface in the **graphics** package for directly rotating text about its center which is achievable however by fiddling with adj and srt simultaneously.

Graphical parameters col, cex and font can be vectors and will then be applied cyclically to the labels (and extra values will be ignored). NA values of font are replaced by par("font"), and similarly for col.

Labels whose x, y or labels value is NA are omitted from the plot.

What happens when font = 5 (the symbol font) is selected can be both device- and locale-dependent. Most often labels will be interpreted in the Adobe symbol encoding, so e.g. "d" is delta, and "\300" is aleph.

Euro symbol

The Euro symbol may not be available in older fonts. In current versions of Adobe symbol fonts it is character 160, so text(x, y, "\xA0", font = 5) may work. People using Western European locales on Unix-alikes can probably select ISO-8895-15 (Latin-9) which has the Euro as character 165: this can also be used for pdf and postscript. It is '\u20ac' in Unicode, which can be used in UTF-8 locales.

The Euro should be rendered correctly by X11 in UTF-8 locales, but the corresponding single-byte encoding in postscript and pdf will need to be selected as ISOLatin9.enc (and the font will need to contain the Euro glyph, which for example older printers may not).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

See Also

text.formula for the formula method; mtext, title, Hershey for details on Hershey vector fonts, plotmath for details and more examples on mathematical annotation.

Examples

```
plot(-1:1, -1:1, type = "n", xlab = "Re", ylab = "Im")
K <- 16; text(exp(1i * 2 * pi * (1:K) / K), col = 2)

## The following two examples use latin1 characters: these may not
## appear correctly (or be omitted entirely).
plot(1:10, 1:10, main = "text(...) examples\n~~~~~~~~~~~~~~~",
     sub = "R is GNU ©, but not ® ...")
mtext("«Latin-1 accented chars»: éè øØ å<Å æ<Æ", side = 3)
points(c(6,2), c(2,1), pch = 3, cex = 4, col = "red")
text(6, 2, "the text is CENTERED around (x,y) = (6,2) by default",
     cex = .8)
```

```
text(2, 1, "or Left/Bottom - JUSTIFIED at (2,1) by 'adj = c(0,0)'",
     adj = c(0,0))
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
     cex = .75)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))

## Two more latin1 examples
text(5, 10.2,
     "Le français, c'est facile: Règles, Liberté, Egalité, Fraternité...")
text(5, 9.8,
     "Jetz no chli züritüütsch: (noch ein bißchen Zürcher deutsch)")
```

`title` *Plot Annotation*

Description

This function can be used to add labels to a plot. Its first four principal arguments can also be used as arguments in most high-level plotting functions. They must be of type `character` or `expression`. In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, Greek letters, fractions, etc: see plotmath

Usage

```
title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
      line = NA, outer = FALSE, ...)
```

Arguments

`main`	The main title (on top) using font, size (character expansion) and color `par(c("font.main", "cex.main", "col.main"))`.
`sub`	Sub-title (at bottom) using font, size and color `par(c("font.sub", "cex.sub", "col.sub"))`.
`xlab`	X axis label using font, size and color `par(c("font.lab", "cex.lab", "col.lab"))`.
`ylab`	Y axis label, same font attributes as `xlab`.
`line`	specifying a value for `line` overrides the default placement of labels, and places them this many lines outwards from the plot edge.
`outer`	a logical value. If `TRUE`, the titles are placed in the outer margins of the plot.
`...`	further graphical parameters from `par`. Use e.g., `col.main` or `cex.sub` instead of just `col` or `cex`. `adj` controls the justification of the titles. `xpd` can be used to set the clipping region: this defaults to the figure region unless `outer = TRUE`, otherwise the device region and can only be increased. `mgp` controls the default placing of the axis titles.

Details

The labels passed to `title` can be character strings or language objects (names, calls or expressions), or a list containing the string to be plotted, and a selection of the optional modifying graphical parameters `cex=`, `col=` and `font=`. Other objects will be coerced by `as.graphicsAnnot`.

The position of `main` defaults to being vertically centered in (outer) margin 3 and justified horizontally according to `par("adj")` on the plot region (device region for `outer = TRUE`).

The positions of `xlab`, `ylab` and `sub` are `line` (default for `xlab` and `ylab` being `par("mgp")[1]` and increased by 1 for `sub`) lines (of height `par("mex")`) into the appropriate margin, justified in the text direction according to `par("adj")` on the plot/device region.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`mtext`, `text`; `plotmath` for details on mathematical annotation.

Examples

```
plot(cars, main = "") # here, could use main directly
title(main = "Stopping Distance versus Speed")

plot(cars, main = "")
title(main = list("Stopping Distance versus Speed", cex = 1.5,
                  col = "red", font = 3))

## Specifying "..." :
plot(1, col.axis = "sky blue", col.lab = "thistle")
title("Main Title", sub = "subtitle",
      cex.main = 2,   font.main= 4, col.main= "blue",
      cex.sub = 0.75, font.sub = 3, col.sub = "red")


x <- seq(-4, 4, length.out = 101)
y <- cbind(sin(x), cos(x))
matplot(x, y, type = "l", xaxt = "n",
        main = expression(paste(plain(sin) * phi, "  and  ",
                                plain(cos) * phi)),
        ylab = expression("sin" * phi, "cos" * phi), # only 1st is taken
        xlab = expression(paste("Phase Angle ", phi)),
        col.main = "blue")
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
     labels = expression(-pi, -pi/2, 0, pi/2, pi))
abline(h = 0, v = pi/2 * c(-1,1), lty = 2, lwd = .1, col = "gray70")
```

`units` *Graphical Units*

Description

xinch and yinch convert the specified number of inches given as their arguments into the correct units for plotting with graphics functions. Usually, this only makes sense when normal coordinates are used, i.e., *no* log scale (see the log argument to par).

xyinch does the same for a pair of numbers xy, simultaneously.

Usage

```
xinch(x = 1, warn.log = TRUE)
yinch(y = 1, warn.log = TRUE)
xyinch(xy = 1, warn.log = TRUE)
```

Arguments

x, y	numeric vector
xy	numeric of length 1 or 2.
warn.log	logical; if TRUE, a warning is printed in case of active log scale.

Examples

```
all(c(xinch(), yinch()) == xyinch()) # TRUE
xyinch()
xyinch #- to see that is really   delta{"usr"} / "pin"

## plot labels offset 0.12 inches to the right
## of plotted symbols in a plot
with(mtcars, {
    plot(mpg, disp, pch = 19, main = "Motor Trend Cars")
    text(mpg + xinch(0.12), disp, row.names(mtcars),
         adj = 0, cex = .7, col = "blue")
    })
```

---

xspline *Draw an X-spline*

Description

Draw an X-spline, a curve drawn relative to control points.

Usage

```
xspline(x, y = NULL, shape = 0, open = TRUE, repEnds = TRUE,
        draw = TRUE, border = par("fg"), col = NA, ...)
```

Arguments

x, y	vectors containing the coordinates of the vertices of the polygon. See xy.coords for alternatives.
shape	A numeric vector of values between -1 and 1, which control the shape of the spline relative to the control points.
open	A logical value indicating whether the spline is an open or a closed shape.

repEnds	For open X-splines, a logical value indicating whether the first and last control points should be replicated for drawing the curve. Ignored for closed X-splines.
draw	logical: should the X-spline be drawn? If false, a set of line segments to draw the curve is returned, and nothing is drawn.
border	the color to draw the curve. Use border = NA to omit borders.
col	the color for filling the shape. The default, NA, is to leave unfilled.
...	graphical parameters such as lty, xpd, lend, ljoin and lmitre can be given as arguments.

Details

An X-spline is a line drawn relative to control points. For each control point, the line may pass through (interpolate) the control point or it may only approach (approximate) the control point; the behaviour is determined by a shape parameter for each control point.

If the shape parameter is greater than zero, the spline approximates the control points (and is very similar to a cubic B-spline when the shape is 1). If the shape parameter is less than zero, the spline interpolates the control points (and is very similar to a Catmull-Rom spline when the shape is -1). If the shape parameter is 0, the spline forms a sharp corner at that control point.

For open X-splines, the start and end control points must have a shape of 0 (and non-zero values are silently converted to zero).

For open X-splines, by default the start and end control points are replicated before the curve is drawn. A curve is drawn between (interpolating or approximating) the second and third of each set of four control points, so this default behaviour ensures that the resulting curve starts at the first control point you have specified and ends at the last control point. The default behaviour can be turned off via the repEnds argument.

Value

If draw = TRUE, NULL otherwise a list with elements x and y which could be passed to lines, polygon and so on.

Invisible in both cases.

Note

Two-dimensional splines need to be created in an isotropic coordinate system. Device coordinates are used (with an anisotropy correction if needed.)

References

Blanc, C. and Schlick, C. (1995), *X-splines : A Spline Model Designed for the End User*, in *Proceedings of SIGGRAPH 95*, pp. 377–386. https://dept-info.labri.fr/~schlick/DOC/sig1.html

See Also

polygon.

par for how to specify colors.

Examples

```
## based on examples in ?grid.xspline

xsplineTest <- function(s, open = TRUE,
                        x = c(1,1,3,3)/4,
                        y = c(1,3,3,1)/4, ...) {
    plot(c(0,1), c(0,1), type = "n", axes = FALSE, xlab = "", ylab = "")
    points(x, y, pch = 19)
    xspline(x, y, s, open, ...)
    text(x+0.05*c(-1,-1,1,1), y+0.05*c(-1,1,1,-1), s)
}
op <- par(mfrow = c(3,3), mar = rep(0,4), oma = c(0,0,2,0))
xsplineTest(c(0, -1, -1, 0))
xsplineTest(c(0, -1,  0, 0))
xsplineTest(c(0, -1,  1, 0))
xsplineTest(c(0,  0, -1, 0))
xsplineTest(c(0,  0,  0, 0))
xsplineTest(c(0,  0,  1, 0))
xsplineTest(c(0,  1, -1, 0))
xsplineTest(c(0,  1,  0, 0))
xsplineTest(c(0,  1,  1, 0))
title("Open X-splines", outer = TRUE)

par(mfrow = c(3,3), mar = rep(0,4), oma = c(0,0,2,0))
xsplineTest(c(0, -1, -1, 0), FALSE, col = "grey80")
xsplineTest(c(0, -1,  0, 0), FALSE, col = "grey80")
xsplineTest(c(0, -1,  1, 0), FALSE, col = "grey80")
xsplineTest(c(0,  0, -1, 0), FALSE, col = "grey80")
xsplineTest(c(0,  0,  0, 0), FALSE, col = "grey80")
xsplineTest(c(0,  0,  1, 0), FALSE, col = "grey80")
xsplineTest(c(0,  1, -1, 0), FALSE, col = "grey80")
xsplineTest(c(0,  1,  0, 0), FALSE, col = "grey80")
xsplineTest(c(0,  1,  1, 0), FALSE, col = "grey80")
title("Closed X-splines", outer = TRUE)

par(op)

x <- sort(stats::rnorm(5))
y <- sort(stats::rnorm(5))
plot(x, y, pch = 19)
res <- xspline(x, y, 1, draw = FALSE)
lines(res)
## the end points may be very close together,
## so use last few for direction
nr <- length(res$x)
arrows(res$x[1], res$y[1], res$x[4], res$y[4], code = 1, length = 0.1)
arrows(res$x[nr-3], res$y[nr-3], res$x[nr], res$y[nr], code = 2, length = 0.1)
```