Package 'latticeExtra'

December 19, 2019

Version 0.6-29

Date 2019-12-18

Title Extra Graphical Utilities Based on Lattice

Author Deepayan Sarkar <deepayan.sarkar@r-project.org>, Felix Andrews <felix@nfrac.org>

Maintainer Deepayan Sarkar <deepayan.sarkar@r-project.org>

Description Building on the infrastructure provided by the lattice package, this package provides several new high-level functions and methods, as well as additional utilities such as panel and axis annotation functions.

Depends R (>= 3.6.0), lattice

Imports grid, stats, utils, grDevices, png, jpeg, RColorBrewer

Suggests maps, mapproj, deldir, tripack, quantreg, zoo, MASS, mgcv

URL http://latticeextra.r-forge.r-project.org/

LazyLoad yes

LazyData yes

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-19 13:50:05 UTC

R topics documented:

ancestry 2
as.layer 3
biocAccess 5
c.trellis 6
combineLimits 9
custom.theme 11
dendrogramGrob 12
doubleYScale 14

EastAuClimate . . . 16
ecdfplot . . . 19
ggplot2like.theme . . . 21
gvhd10 . . . 23
horizonplot . . . 24
layer . . . 28
mapplot . . . 32
marginal.plot . . . 35
panel.2dsmoother . . . 37
panel.3dmisc . . . 38
panel.ellipse . . . 41
panel.key . . . 42
panel.lmlineq . . . 43
panel.qqmath.tails . . . 46
panel.quantile . . . 47
panel.scaleArrow . . . 48
panel.segplot . . . 50
panel.smoother . . . 51
panel.tskernel . . . 53
panel.voronoi . . . 55
panel.xblocks . . . 57
panel.xyarea . . . 59
panel.xyimage . . . 61
postdoc . . . 62
resizePanels . . . 63
rootogram . . . 64
scale.components . . . 68
SeatacWeather . . . 69
segplot . . . 70
theEconomist.theme . . . 72
tileplot . . . 74
USAge . . . 76
USCancerRates . . . 77
useOuterStrips . . . 78
xyplot.stl . . . 80

Index **82**

`ancestry` *Modal ancestry by County according to US 2000 Census*

Description

This data set records the population and the three most frequently reported ancestries by US county, according to the 2000 census.

Usage

```
data(ancestry)
```

Format

A data frame with 3219 observations on the following 5 variables.

`county` A factor. An attempt has been made to make the levels look similar to the county names used in the `maps` package.

`population` a numeric vector

`top` a character vector

`second` a character vector

`third` a character vector

Source

U.S. Census Bureau. The ancestry data were extracted from Summary File 3, available from

http://www.census.gov/census2000/sumfile3.html

which is based on the 'long form' questionnaire (asked to 1 in 6 households surveyed).

References

http://www.census.gov/prod/cen2000/doc/sf3.pdf

See Also

`mapplot`, for examples.

---

as.layer *Overlay panels of Trellis plots on same or different scales*

Description

Allows overlaying of Trellis plots, drawn on the same scales or on different scales in each of the x and y dimensions. There are options for custom axes and graphical styles.

Usage

```
as.layer(x, ...)

## S3 method for class 'trellis'
as.layer(x, x.same = TRUE, y.same = TRUE,
         axes = c(if (!x.same) "x", if (!y.same) "y"), opposite = TRUE,
         outside = FALSE, theme = x$par.settings, ...)
```

Arguments

x	a trellis object.
x.same	retains the existing panel x scale for the new layer, rather than using the layer's native x scale.
y.same	retains the existing panel y scale.
axes	which of the axes to draw (NULL for neither). Axes might not be drawn anyway, such as if scales$draw == FALSE.
opposite	whether to draw axes on the opposite side to normal: that is, the top and/or right sides rather than bottom and/or left. May be a vector of length 2 to specify for x and y axes separately.
outside	whether to draw the axes outside the plot region. Note that space for outside axes will not be allocated automatically. May be a vector of length 2 to specify for x and y axes separately.
theme	passed to layer.
...	passed to layer: typically the style argument would be specified.

Details

Panels from the trellis object x will be drawn in the corresponding panel of another trellis object, so packet numbers match (see examples).

Axis setting are taken from the trellis object x, so most scales arguments such as draw, at, labels etc will carry over to the overlaid axes. Only the main axis settings are used (i.e. left or bottom), even when opposite = TRUE.

Currently, outside top axes will be drawn in the strip if there are strips.

Value

an updated trellis object.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

doubleYScale, layer, panel.axis

Examples

```
b1 <- barley
b2 <- barley
b2$yield <- b2$yield + 10

## panels are matched up by packet number
dotplot(variety ~ yield | site * year, b1) +
  as.layer(dotplot(variety ~ yield | site * year, b2, col = "red"))
```

```
## which gives the same result as:
dotplot(variety ~ yield | site * year, data = b1, subscripts = TRUE) +
  layer(panel.dotplot(yield[subscripts], variety[subscripts], col = "red"),
        data = b2)


## example with all same scales (the default):
xyplot(fdeaths ~ mdeaths) +
  as.layer(xyplot(fdeaths ~ mdeaths, col = 2, subset = ldeaths > 2000))

## same x scales, different y scales:
xyplot(fdeaths ~ mdeaths) +
  as.layer(bwplot(~ mdeaths, box.ratio = 0.2), y.same = FALSE)

## same y scales, different x scales:
xyplot(fdeaths ~ mdeaths) +
  as.layer(bwplot(fdeaths ~ factor(mdeaths*0), box.ratio = 0.2), x.same = FALSE)

## as.layer() is called automatically if two plots are added:
histogram(~ ldeaths, type = "density") + densityplot(~ ldeaths, lwd = 3)

## applying one panel layer to several panels of another object
xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species,
       data = iris, scales = "free") +
   as.layer(levelplot(volcano), x.same = FALSE, y.same = FALSE, under = TRUE)
```

`biocAccess` *Hourly access attempts to Bioconductor website*

Description

This data set records the hourly number of access attempts to the Bioconductor website (http://www.bioconductor.org) during January through May of 2007. The counts are essentially an aggregation of the number of entries in the access log.

Usage

```
data(biocAccess)
```

Format

A data frame with 3623 observations on the following 7 variables.

`counts` the number of access attempts

`day` the day of the month

`month` a factor with levels `Jan`, `Feb`, ..., `Dec`

`year` the year (all 2007)

`hour` hour of the day, a numeric vector

`weekday` a factor with levels `Monday`, `Tuesday`, ..., `Sunday`

`time` a POSIXt representation of the start of the hour

Examples

```
data(biocAccess)
xyplot(stl(ts(biocAccess$counts[1:(24 * 30)], frequency = 24), "periodic"))
```

`c.trellis` *Merge trellis objects, using same or different scales*

Description

Combine the panels of multiple trellis objects into one.

Usage

```
## S3 method for class 'trellis'
c(..., x.same = NA, y.same = NA,
    layout = NULL, merge.legends = FALSE, recursive = FALSE)

xyplot.list(x, data = NULL, ..., FUN = xyplot,
    y.same = TRUE, x.same = NA, layout = NULL,
    merge.legends = FALSE)
```

Arguments

`...`	two or more trellis objects. If these are named arguments, the names will be used in the corresponding panel strips.
`x.same`	if `TRUE`, set the x scale relation to `"same"` and recalculate panel limits using data from all panels. Otherwise, the x scales in each panel will be as they were in the original objects (so in general not the same), the default behaviour.
`y.same`	as above, for y scales. Note that `xyplot.list` defaults to same y scales. Set to `NA` to leave them alone.
`layout`	value for `layout` of the new plot; see `xyplot`.
`merge.legends`	to keep keys or legends from all plots, not just the first. If multiple legends share the same `"space"`, they are packed together horizontally or vertically.
`recursive`	for consistency with the generic method, ignored.
`x`	a `list` of objects to plot individually, and then be combined into one final plot.
`FUN, data`	a lattice plot function, to be called on each element of the list `x`, along with `data` and `...`

Details

This mechanism attempts to merge the panels from multiple trellis objects into one. The same effect could generally be achieved by either a custom panel function (where the display depends on `packet.number()`), or using `print.trellis` to display multiple trellis objects. However, in some cases it is more convenient to use `c()`. Furthermore, it can be useful to maintain the display

as a standard lattice display, rather than a composite using print.trellis, to simplify further interaction.

Many properties of the display, such as titles, axis settings and aspect ratio will be taken from the first object only.

Note that combining panels from different types of plots does not really fit the trellis model. Some features of the plot may not work as expected. In particular, some work may be needed to show or hide scales on selected panels. An example is given below.

Any trellis object with more than one conditioning variable will be "flattened" to one dimension, eliminating the multi-variate conditioning structure.

Value

a new trellis object.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

marginal.plot was the original motivating application, print.trellis, update.trellis, trellis.object

Examples

```
## Combine different types of plots.
c(wireframe(volcano), contourplot(volcano))

## Merging levelplot with xyplot
levObj <- levelplot(prop.table(WorldPhones, 1) * 100)
xyObj <- xyplot(Phones ~ Year, data.frame(Phones = rowSums(WorldPhones),
      Year = row.names(WorldPhones)), type="b", ylim = c(0, 150000))
## NOTE: prepanel.levelplot (from first object) is used for entire plot.
cObj <- c(levObj, xyObj, layout = 1:2)
update(cObj, scales = list(y = list(rot = 0)),
       ylab = c("proportional distribution", "number of phones"))

## Combine two xyplots.
sepals <- xyplot(Sepal.Length ~ Sepal.Width, iris, groups = Species,
    xlab = "Width", ylab = "Height")
petals <- xyplot(Petal.Length ~ Petal.Width, iris, groups = Species)
c(Sepals = sepals, Petals = petals)

## Force same scales (re-calculate panel limits from merged data):
c(Sepals = sepals, Petals = petals, x.same = TRUE, y.same = TRUE)

## Or - create xyplots from a list of formulas
xyplot.list(list(Sepals = Sepal.Length ~ Sepal.Width,
                 Petals = Petal.Length ~ Petal.Width),
            data = iris, groups = Species, x.same = TRUE,
            xlab = "Width", ylab = "Height")
```

```
## Create histograms from a list of objects, and merge them.
xyplot.list(iris, FUN = histogram)

## Create cumulative distribution plots from a list of objects
xyplot.list(iris[1:4], FUN = qqmath, groups = iris$Species,
            auto.key = TRUE)

## Display a table as both frequencies and proportions:
data(postdoc)
## remove last row (containing totals)
postdoc <- postdoc[1:(nrow(postdoc)-1),]
pdprops <- barchart(prop.table(postdoc, margin = 1),
                    auto.key = list(adj = 1))
pdmargin <- barchart(margin.table(postdoc, 1))
pdboth <- c(pdprops, pdmargin)
update(pdboth, xlab = c("Proportion", "Freq"))

## Conditioned 'quakes' plot combined with histogram.
qua <- xyplot(lat ~ long | equal.count(depth, 3), quakes,
    aspect = "iso", pch = ".", cex = 2, xlab = NULL, ylab = NULL)
qua <- c(qua, depth = histogram(quakes$depth), layout = c(4, 1))
## suppress scales on the first 3 panels
update(qua, scales = list(at = list(NULL, NULL, NULL, NA),
                          y = list(draw = FALSE)))

## Demonstrate merging of legends and par.settings.
## Note that par.settings can conflict, thus need col.line=...
mypoints <-
    xyplot(1:10 ~ 1:10, groups = factor(rep(1:2, each = 5)),
        par.settings = simpleTheme(pch = 16), auto.key = TRUE)
mylines <-
    xyplot(1:10 ~ 1:10, groups = factor(rep(1:5, each = 2)),
        type = "l", par.settings = simpleTheme(col.line = 1:5),
        auto.key = list(lines = TRUE, points = FALSE, columns = 5))
c(mypoints, mylines)

## Visualise statistical and spatial distributions
## (advanced!)
library(maps)
vars <- as.data.frame(state.x77)
StateName <- tolower(state.name)
form <- StateName ~ Population + Income + Illiteracy +
   `Life Exp` + Murder + `HS Grad` + Frost + sqrt(Area)
## construct independent maps of each variable
statemap <- map("state", plot = FALSE, fill = TRUE)
colkey <- draw.colorkey(list(col = heat.colors(100), at = 0:100,
    labels = list(labels = c("min","max"), at = c(0,100))))
panel.mapplot.each <- function(x, breaks, ...)
    panel.mapplot(x = x, breaks = quantile(x), ...)
vmaps <- mapplot(form, vars, map = statemap, colramp = heat.colors,
    panel = panel.mapplot.each, colorkey = FALSE,
    legend = list(right = list(fun = colkey)), xlab = NULL)
```

```
## construct independent densityplots of each variable
vdens <- densityplot(form[-2], vars, outer = TRUE, cut = 0,
    scales = list(relation = "free"), ylim = c(0, NA),
    cex = 0.5, ref = TRUE) +
  layer(panel.axis("top", half = FALSE, text.cex = 0.7))
## combine panels from both plots
combo <- c(vmaps, vdens)
## rearrange in pairs
n <- length(vars)
npairs <- rep(1:n, each = 2) + c(0, n)
update(combo[npairs], scales = list(draw = FALSE),
    layout = c(4, 4), between = list(x = c(0, 0.5), y = 0.5))
```

`combineLimits` *Combine axis limits across margins*

Description

Modifies a `"trellis"` object with `"free"` scales so that panel limits are extended to be the same across selected conditioning variables (typically rows and columns).

Usage

```
combineLimits(x, margin.x = 2L, margin.y = 1L,
              extend = TRUE, adjust.labels = TRUE)
```

Arguments

`x`	An object of class `"trellis"`.
`margin.x`	Integer vector specifying which conditioning variables to combine the x-axis limits over. Defaults to the second conditioning variable (rows in the default layout); that is, the limit of each packet is extended to include the limits of all other packets obtained by varying the level of the second conditioning variable (row). More than one variable can be specified; for example, `margin.x = c(1,2)` would ensure that limits are extended to include all levels in both the first and second conditioning variables. In case there is a third conditioning variable, this would have the effect of per-page x-axis limits with the default layout.
`margin.y`	Integer vector specifying which conditioning variables to combine the x-axis limits over. Similar to `margin.x`, but defaults to the first conditioning variable (columns in the default layout).
`extend`	Logical flag indicating whether the limits should be extended after being combined. Usually a good idea.
`adjust.labels`	Logical flag indicating whether labels should be removed from all but the boundaries. This may give misleading plots with non-default layouts.

Details

combineLimits is useful mainly for plots with two conditioning variables with the default layout (columns and rows correspond to the first and second conditioning variables), when per-row and per-column limits are desired.

The lattice approach does not tie levels of the conditioning variables to the plot layout, so it is possible that all panels in a row (or column) do not represent the same level. It should be noted that combineLimits actually combines limits across levels, and not across rows and columns. Results are likely to be misleading unless the default layout is used.

Value

An object of class "trellis"; essentially the same as x, but with certain properties modified.

Note

Does not work for all "trellis" objects. In particular, log-scales do not yet work. Fancy layouts with skip-ped panels and unusual packet-to-panel mappings will probably also not work.

Author(s)

Deepayan Sarkar

See Also

Lattice, xyplot

Examples

```
data(Cars93, package = "MASS")

## FIXME: log scales don't yet work

pcars <-
    xyplot(Price ~ EngineSize | reorder(AirBags, Price) + Cylinders,
           data = Cars93,
           subset = Cylinders != "rotary" & Cylinders != "5",
           scales = list(relation = "free",
                         y = list(log = FALSE, tick.number = 3, rot = 0)),
           xlab = "Engine Size (litres)",
           ylab = "Average Price (1000 USD)",
           as.table = TRUE)

combineLimits(pcars)

useOuterStrips(combineLimits(update(pcars, grid = TRUE),
                             margin.x = c(1, 2), adjust.labels = FALSE))

useOuterStrips(combineLimits(update(pcars, grid = TRUE)))
```

custom.theme *Create a lattice theme based on specified colors*

Description

Creates a lattice theme given a few colors. Non-color settings are not included. The colors are typically used to define the standard grouping (superposition) colors, and the first color is used for ungrouped displays.

Usage

```
custom.theme(symbol = RColorBrewer::brewer.pal(n = 8, name = "Dark2"),
             fill = RColorBrewer::brewer.pal(n = 12, name = "Set3"),
             region = RColorBrewer::brewer.pal(n = 11, name = "Spectral"),
             reference = "#e8e8e8",
             bg = "transparent",
             fg = "black",
             ...)

## different defaults ("Set1", "Accent", "RdBu"):
custom.theme.2(...)
```

Arguments

symbol	a vector of symbol colors.
fill	a vector of fill colors (for barcharts, etc.)
region	a vector of colors that is used to define a continuous color gradient using colorRampPalette
reference	a color for reference lines and such
bg	a background color
fg	a foreground color, primarily for annotation
...	further arguments passed to simpleTheme and used to modify the theme.

Value

A list that can be supplied to trellis.par.get or as the theme argument to trellis.device.

Author(s)

Deepayan Sarkar

Examples

```
set.seed(0)

## create a plot to demonstrate graphical settings
obj <-
xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width, iris,
       type = c("p", "r"), jitter.x = TRUE, jitter.y = TRUE, factor = 5,
       auto.key = list(lines = TRUE, rectangles = TRUE))
obj <- update(obj, legend = list(right =
         list(fun = "draw.colorkey", args = list(list(at = 0:100)))))


## draw with default theme
obj

## draw with custom.theme()
update(obj, par.settings = custom.theme())

## create a theme with paired colours, filled points, etc
update(obj, par.settings =
  custom.theme(symbol = RColorBrewer::brewer.pal(12, "Paired"),
               fill = RColorBrewer::brewer.pal(12, "Paired"),
               region = RColorBrewer::brewer.pal(9, "Blues"),
               bg = "grey90", fg = "grey20", pch = 16))

## draw with custom.theme.2()
update(obj, par.settings = custom.theme.2())
```

---

dendrogramGrob *Create a Grob Representing a Dendrogram*

Description

This function creates a grob (a grid graphics object) that can be manipulated as such. In particular, it can be used as a legend in a lattice display like `levelplot` to form heatmaps.

Usage

```
dendrogramGrob(x, ord = order.dendrogram(x),
               side = c("right", "top"),
               add = list(), size = 5, size.add = 1,
               type = c("rectangle", "triangle"),
               ...)
```

Arguments

x	An object of class "dendrogram". See dendrogram for details

ord	A vector of integer indices giving the order in which the terminal leaves are to be plotted. If this is not the same as order.dendrogram(x), then the leaves may not cluster together and branches of the dendrogram may intersect.
side	Intended position of the dendrogram when added in a heatmap. Currently allowed positions are "right" and "top".
add	Additional annotation. Currently, it is only possible to add one or more rows of rectangles at the base of the dendrogram. See details below.
size	Total height of the dendrogram in "lines" (see unit)
size.add	Size of each additional row, also in "lines"
type	Whether a child node is joined to its parent directly with a straight line ("triangle") or as a "stair" with two lines ("rectangle")
...	Extra arguments. Currently ignored.

Details

The add argument can be used for additional annotation at the base of the dendrogram. It should be a list with one component for each row, with names specifying the type of annotation and components specifying the contents. Currently, the only supported name is "rect" (which can be repeated), producing rectangles. The components in such a case is a list of graphical parameters, possibly vectorized, that are passed on to gpar.

Value

An object of class "grob"

Author(s)

Deepayan Sarkar <deepayan.sarkar@r-project.org>

See Also

heatmap, levelplot

Examples

```
data(mtcars)
x  <- t(as.matrix(scale(mtcars)))
dd.row <- as.dendrogram(hclust(dist(x)))
row.ord <- order.dendrogram(dd.row)

dd.col <- as.dendrogram(hclust(dist(t(x))))
col.ord <- order.dendrogram(dd.col)

library(lattice)

levelplot(x[row.ord, col.ord],
          aspect = "fill",
          scales = list(x = list(rot = 90)),
```

```
          colorkey = list(space = "left"),
          legend =
          list(right =
               list(fun = dendrogramGrob,
                    args =
                    list(x = dd.col, ord = col.ord,
                         side = "right",
                         size = 10)),
               top =
               list(fun = dendrogramGrob,
                    args =
                    list(x = dd.row,
                         side = "top",
                         type = "triangle"))))


## Version with unsorted rows.

## For illustration, MPG ploted again at foot of dendrogram

mpg <- x["mpg", col.ord]
mpg.col <- level.colors(mpg, at = do.breaks(range(mpg), 10),
                        col.regions = terrain.colors)

levelplot(x[, col.ord],
          aspect = "iso",
          scales = list(x = list(rot = 90)),
          colorkey = FALSE,
          legend =
          list(right =
               list(fun = dendrogramGrob,
                    args =
                    list(x = dd.col, ord = col.ord,
                         side = "right",
                         add = list(rect = list(fill = mpg.col)),
                         size = 10)),
               top =
               list(fun = dendrogramGrob,
                    args =
                    list(x = dd.row, ord = sort(row.ord),
                         side = "top", size = 10,
                         type = "triangle"))))
```

---

doubleYScale *Draw two plot series with different y scales*

Description

Overplot two trellis objects with different y scales, optionally in different styles, adding a second y axis, and/or a second y axis label.

Note: drawing plots with multiple scales is often a bad idea as it can be misleading.

Usage

```
doubleYScale(obj1, obj2, use.style = TRUE,
             style1 = if (use.style) 1, style2 = if (use.style) 2,
             add.axis = TRUE, add.ylab2 = FALSE,
             text = NULL, auto.key = if (!is.null(text))
               list(text, points = points, lines = lines, ...),
             points = FALSE, lines = TRUE, ..., under = FALSE)
```

Arguments

`obj1, obj2`	trellis objects. Note that most settings, like main/sub/legend/etc are taken only from `obj1`; only the panel, axis and ylab are taken from obj2.
`use.style, style1, style2`	`style1` and `style2` give the 'group number' for obj1 and obj2 respectively. The style is taken from these indices into the values of `trellis.par.get("superpose.line")`. Therefore these should be integers between 1 and 6; a value of 0 or NULL can be given to leave the default settings. These will also be applied to the y-axes and ylab, if relevant. `use.style` simply changes the defaults of the style arguments.
`add.axis`	if TRUE, draw a second y axis (for the `obj2` series) on the right side of the plot.
`add.ylab2`	if TRUE, draw a second y axis label (from `obj2$ylab`) on the right side of the plot. Note, this will replace any existing key or legend on the right side, i.e. with `space = "right"`.
`text, auto.key, points, lines, ...`	if non-NULL, add a `key` to the display, using entries named by `text`. Further arguments are passed on to `simpleKey` at plot time.
`under`	if TRUE, draw `obj2` under `obj1`.

Details

Panels from the trellis object `obj2` will be drawn in the corresponding panel of `obj1`.

Axis settings are taken from the trellis objects, so most `scales` arguments such as `draw`, `at`, `labels` etc from `obj2` will carry over to the second y axis.

Value

a merged trellis object.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`as.layer`

Examples

```
set.seed(1)
foo <- list(x = 1:100, y = cumsum(rnorm(100)))
## show original data
xyplot(y + y^2 ~ x, foo, type = "l")
## construct separate plots for each series
obj1 <- xyplot(y ~ x, foo, type = "l")
obj2 <- xyplot(y^2 ~ x, foo, type = "l")
## simple case: no axis for the overlaid plot
doubleYScale(obj1, obj2, add.axis = FALSE)
## draw second y axis
doubleYScale(obj1, obj2)
## ...with second ylab
doubleYScale(obj1, obj2, add.ylab2 = TRUE)
## ...or with a key
doubleYScale(obj1, obj2, text = c("obj1", "obj2"))
## ...with custom styles
update(doubleYScale(obj1, obj2, text = c("obj1", "obj2")),
  par.settings = simpleTheme(col = c('red','black'), lty = 1:2))

## different plot types
x <- rnorm(60)
doubleYScale(histogram(x), densityplot(x), use.style = FALSE)
## (but see ?as.layer for a better way to do this)

## multi-panel example
## a variant of Figure 5.13 from Sarkar (2008)
## http://lmdvr.r-forge.r-project.org/figures/figures.html?chapter=05;figure=05_13
data(SeatacWeather)
temp <- xyplot(min.temp + max.temp ~ day | month,
               data = SeatacWeather, type = "l", layout = c(3, 1))
rain <- xyplot(precip ~ day | month, data = SeatacWeather, type = "h")

doubleYScale(temp, rain, style1 = 0, style2 = 3, add.ylab2 = TRUE,
   text = c("min. T", "max. T", "rain"), columns = 3)

## re-plot with different styles
update(trellis.last.object(),
   par.settings = simpleTheme(col = c("black", "red", "blue")))
```

`EastAuClimate` *Climate of the East Coast of Australia*

Description

A set of climate statistics for 16 coastal locations along Eastern Australia. These sites were chosen to be approximately equally spaced to cover the whole eastern coast of Australia. For each site, climate statistics were calculated for the standard 30-year period 1971-2000. Only sites with nearly-complete data were chosen.

Usage

```
data(EastAuClimate)
```

Format

A data frame with the following 10 variables and 5 items of metadata for each of 16 sites.

`SummerMaxTemp` average daily maximum air temperature (degrees C) in February.

`SummerMinTemp` average daily minimum air temperature (degrees C) in February.

`WinterMaxTemp` average daily maximum air temperature (degrees C) in July.

`WinterMinTemp` average daily minimum air temperature (degrees C) in July.

`SummerRain` median total precipitation in February (mm/month).

`WinterRain` median total precipitation in July (mm/month).

`MeanAnnRain` average total amount of precipitation recorded in a year (mm/year).

`RainDays` average number of days in a year with at least 1 mm of precipitation.

`ClearDays` average number of clear days in a year. This statistic is derived from cloud cover observations, which are measured in oktas (eighths). A clear day is recorded when the mean of the 9 am and 3 pm cloud observations is less than or equal to 2 oktas.

`CloudyDays` average number of clear days in a year. A cloudy day is recorded when the mean of the 9 am and 3 pm cloud observations is greater than or equal to 6 oktas.

`ID` BOM Site number.

`Latitude` Site latitude (degrees North).

`Longitude` Site longitude (degrees East).

`Elevation` Site elevation (m).

`State` Australian state: TAS = Tasmania, VIC = Victoria, NSW = New South Wales, QLD = Queensland.

The row names of the data frame give the location names. Note: these are not the official names of the climate stations.

Source

Sites were chosen by hand from maps on the Bureau of Meteorology website. The data were extracted manually from web pages under http://www.bom.gov.au/climate/ and processed to extract a subset of statistics. - by Felix Andrews <felix@nfrac.org>

Bureau of Meteorology, Commonwealth of Australia. Product IDCJCM0026 Prepared at Wed 31 Dec 2008.

Definitions of statistics adapted from http://www.bom.gov.au/climate/cdo/about/about-stats.shtml

Examples

```
data(EastAuClimate)

## Compare the climates of state capital cities
EastAuClimate[c("Hobart", "Melbourne", "Sydney", "Brisbane"),]

## A function to plot maps (a Lattice version of maps::map)
lmap <-
   function(database = "world", regions = ".", exact = FALSE,
            boundary = TRUE, interior = TRUE, projection = "",
            parameters = NULL, orientation = NULL,
            aspect = "iso", type = "l",
            par.settings = list(axis.line = list(col = "transparent")),
            xlab = NULL, ylab = NULL, ...)
{
   theMap <- map(database, regions, exact = exact,
                 boundary = boundary, interior = interior,
                 projection = projection, parameters = parameters,
                 orientation = orientation, plot = FALSE)
   xyplot(y ~ x, theMap, type = type, aspect = aspect,
          par.settings = par.settings, xlab = xlab, ylab = ylab,
          default.scales = list(draw = FALSE), ...)
}

## Plot the sites on a map of Australia
if (require("maps")) {
  lmap(regions = c("Australia", "Australia:Tasmania"),
       exact = TRUE, projection = "rectangular",
       parameters = 150, xlim = c(130, 170),
       panel = function(...) {
          panel.xyplot(...)
          with(EastAuClimate, {
            panel.points(Longitude, Latitude, pch = 16)
            txt <- row.names(EastAuClimate)
            i <- c(3, 4)
            panel.text(Longitude[ i], Latitude[ i], txt[ i], pos = 2)
            panel.text(Longitude[-i], Latitude[-i], txt[-i], pos = 4)
          })
       })
}

## Average daily maximum temperature in July (Winter).
xyplot(WinterMaxTemp ~ Latitude, EastAuClimate, aspect = "xy",
       type = c("p", "a"), ylab = "Temperature (degrees C)")

## (Make a factor with levels in order - by coastal location)
siteNames <- factor(row.names(EastAuClimate),
           levels = row.names(EastAuClimate))
## Plot temperature ranges (as bars), color-coded by RainDays
segplot(siteNames ~ WinterMinTemp + SummerMaxTemp, EastAuClimate,
        level = RainDays, sub = "Color scale: number of rainy days per year",
        xlab = "Temperature (degrees C)",
```

```
        main = paste("Typical temperature range and wetness",
           "of coastal Australian cities", sep = "\n"))

## Show Winter and Summer temperature ranges separately
segplot(Latitude ~ WinterMinTemp + SummerMaxTemp, EastAuClimate,
   main = "Average daily temperature ranges \n of coastal Australian sites",
   ylab = "Latitude", xlab = "Temperature (degrees C)",
   par.settings = simpleTheme(lwd = 3, alpha = 0.5),
   key = list(text = list(c("July (Winter)", "February (Summer)")),
              lines = list(col = c("blue", "red"))),
   panel = function(x, y, z, ..., col) {
      with(EastAuClimate, {
         panel.segplot(WinterMinTemp, WinterMaxTemp, z, ..., col = "blue")
         panel.segplot(SummerMinTemp, SummerMaxTemp, z, ..., col = "red")
      })
   })

## Northern sites have Summer-dominated rainfall;
## Southern sites have Winter-dominated rainfall.
xyplot(SummerRain + WinterRain ~ Latitude, EastAuClimate,
       type = c("p", "a"), auto.key = list(lines = TRUE),
       ylab = "Rainfall (mm / month)")

## Clear days are most frequent in the mid latitudes.
xyplot(RainDays + CloudyDays + ClearDays ~ Latitude, EastAuClimate,
       type = c("p", "a"), auto.key = list(lines = TRUE),
       ylab = "Days per year")
```

---

ecdfplot *Trellis Displays of Empirical CDF*

Description

Conditional displays of Empirical Cumulative Distribution Functions

Usage

```
ecdfplot(x, data, ...)

## S3 method for class 'formula'
ecdfplot(x, data,
         prepanel = "prepanel.ecdfplot",
         panel = "panel.ecdfplot",
         ylab,
         ...)
## S3 method for class 'numeric'
ecdfplot(x, data = NULL, xlab, ...)

prepanel.ecdfplot(x, f.value = NULL, ...)
```

```
panel.ecdfplot(x, f.value = NULL, type = "s",
               groups = NULL, qtype = 7,
               ref = TRUE,
               ...)
```

Arguments

x	For ecdfplot, x is the object on which method dispatch is carried out. For the "formula" method, x is a formula describing the form of conditioning plot, and has to be of the form ~x, where x is assumed to be a numeric vector. Further conditioning variables are allowed as usual. A similar interpretation holds for x in the "numeric" method as well as prepanel.ecdfplot and panel.ecdfplot.
data	For the "formula" method, a data frame containing values for any variables in the formula, as well as those in groups and subset if applicable.
prepanel, panel	panel and prepanel function used to create the display.
xlab, ylab	axis labels; typically a character string or an expression.
groups	a grouing variable of the same length as x. If specified, ECDF plots are computed for each subset defined by unique values of groups and the resulting functions superposed within each panel.
f.value, qtype	Defines how quantiles are calculated. See panel.qqmath.
ref	logical, whether a reference line should be drawn at 0 and 1
type	how the plot is rendered; see panel.xyplot
...	extra arguments, passed on as appropriate. Standard lattice arguments as well as arguments to panel.ecdfplot can be supplied directly in the high level ecdfplot call.

Value

ecdfplot produces an object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Deepayan Sarkar <deepayan.sarkar@r-project.org>

See Also

qqmath for Quantile plots which are more generally useful, especially when comparing with a theoretical distribution other than uniform. An ECDF plot is essentially a transposed version (i.e., with axes switched) of a uniform quantile plot.

Examples

```
data(singer, package = "lattice")
ecdfplot(~height | voice.part, data = singer)
```

`ggplot2like.theme` *A ggplot2-like theme for Lattice*

Description

A theme for Lattice based on some of the default styles used in the **ggplot2** package by Hadley Wickham. Specifically, parts of the functions `scale_colour_hue`, `scale_colour_gradient` and `theme_gray` were copied. Although superficially similar, the implementation here lacks much of the flexibility of the **ggplot2** functions: see http://had.co.nz/ggplot2/.

Usage

```
ggplot2like(..., n = 6, h = c(0,360) + 15, l = 65, c = 100,
            h.start = 0, direction = 1,
            low = "#3B4FB8", high = "#B71B1A", space = "rgb")

ggplot2like.opts()

axis.grid(side = c("top", "bottom", "left", "right"),
          ..., ticks = c("default", "yes", "no"),
          scales, components, line.col)
```

Arguments

`...`	further arguments passed on to `simpleTheme` to over-ride defaults.
`n`	number of `superpose` styles to generate, with equally spaced hues.
`h, c, l, h.start, direction`	range of hues, starting hue and direction to generate a discrete colour sequence with `hcl`.
`low, high, space`	extreme colors to interpolate with `colorRampPalette` for a continuous color scale.
`side, ticks, scales, components, line.col`	see `axis.default`. Typically `axis.grid` is not called directly so these should not be needed.

Value

`ggplot2like()` produces a list of settings which can be passed as the `par.settings` argument to a high-level Lattice plot, or to `trellis.par.set`. `ggplot2like.opts()` produces a list which can be passed as the `lattice.options` argument to a high-level Lattice plot, or to `lattice.options`.

Author(s)

Felix Andrews <felix@nfrac.org>; copied and adapted from **ggplot2** by Hadley Wickham.

See Also

the **ggplot2** package: http://had.co.nz/ggplot2/.

custom.theme, scale.components

Examples

```
set.seed(0)

## basic theme does not include white grid lines
xyplot(exp(1:10) ~ 1:10, type = "b",
    par.settings = ggplot2like())

## add lines for axis ticks with custom axis function
xyplot(exp(1:10) ~ 1:10, type = "b",
    par.settings = ggplot2like(), axis = axis.grid)

## this can be used together with scale.components
## (minor lines only visible on devices supporting translucency)
xyplot(exp(rnorm(500)) ~ rnorm(500),
    scales = list(y = list(log = TRUE)),
    yscale.components = yscale.components.log10ticks,
    par.settings = ggplot2like(), axis = axis.grid)

## ggplotlike.opts() specifies axis = axis.grid as well as
## xscale.components.subticks / yscale.components.subticks

xyg <- make.groups(
    "group one" = rnorm(80, 1),
    "group two" = rnorm(80, 5),
    "group three" = rnorm(80, 2))
xyg$x <- rev(xyg$data)

## group styles: specify number of equi-spaced hues
xyplot(data ~ x, xyg, groups = which, auto.key = TRUE,
    par.settings = ggplot2like(n = 3),
    lattice.options = ggplot2like.opts()) +
  glayer(panel.smoother(...))

## or set it as the default:
opar <- trellis.par.get()
trellis.par.set(ggplot2like(n = 4, h.start = 180))
oopt <- lattice.options(ggplot2like.opts())

bwplot(voice.part ~ height, data = singer)

histogram(rnorm(100))

barchart(Titanic[,,,"No"], main = "Titanic deaths",
```

```
    layout = c(1, 2), auto.key = list(columns = 2))

## reset
trellis.par.set(opar)
lattice.options(oopt)

## axis.grid and scale.components.subticks can be used alone:
## (again, lines for minor ticks need transculency-support to show up)
xyplot(exp(1:10) ~ 1:10, type = "b",
    lattice.options = ggplot2like.opts(),
    par.settings = list(axis.line = list(col = NA),
        reference.line = list(col = "grey")),
    scales = list(tck = c(0,0)))
```

---

gvhd10 *Flow cytometry data from five samples from a patient*

Description

Flow cytometry data from blood samples taken from a Leukemia patient before and after allogenic bone marrow transplant. The data spans five visits.

Usage

```
data(gvhd10)
```

Format

A data frame with 113896 observations on the following 8 variables.

`FSC.H` forward scatter height values

`SSC.H` side scatter height values

`FL1.H` intensity (height) in the FL1 channel

`FL2.H` intensity (height) in the FL2 channel

`FL3.H` intensity (height) in the FL3 channel

`FL2.A` intensity (area) in the FL2 channel

`FL4.H` intensity (height) in the FL4 channel

`Days` a factor with levels `-6` `0` `6` `13` `20` `27` `34`

Source

http://web.archive.org/web/20070427103026/http://www.ficcs.org/software.html#Data_Files

References

Brinkman, R.R., et al. (2007). High-Content Flow Cytometry and Temporal Data Analysis for Defining a Cellular Signature of Graft-Versus-Host Disease. *Biology of Blood and Marrow Transplantation* **13–6**

Examples

```
## Figure 3.4 from Sarkar (2008)
data(gvhd10)
histogram(~log2(FSC.H) | Days, gvhd10, xlab = "log Forward Scatter",
          type = "density", nint = 50, layout = c(2, 4))
```

---

horizonplot *Plot many time series in parallel*

Description

Plot many time series in parallel by cutting the y range into segments and overplotting them with color representing the magnitude and direction of deviation.

Usage

```
horizonplot(x, data, ...)

## Default S3 method:
horizonplot(x, data = NULL, ...,
            nbands = 3L,
            horizonscale = NA,
            origin = function(y) na.omit(y)[1],
            colorkey = FALSE, legend = NULL,
            panel = panel.horizonplot,
            prepanel = prepanel.horizonplot,
            col.regions = hcl.colors(2 * nbands, palette="RdYlBu"),
            strip = FALSE, strip.left = TRUE,
            par.strip.text = list(cex = 0.6),
            colorkey.digits = 3,
            layout = c(1, NA),
            groups = NULL,
            default.scales =
              list(y = list(relation = "free", axs = "i",
                            draw = FALSE, tick.number = 2)))

panel.horizonplot(x, y, ..., border = NA,
                  nbands = 3L,
                  col.regions = hcl.colors(2 * nbands, palette="RdYlBu"),
                  origin)
```

```
prepanel.horizonplot(x, y, ..., horizonscale = NA,
                     nbands = 3L,
                     origin = function(y) na.omit(y)[1])
```

Arguments

x, y	Argument on which argument dispatch is carried out. Typically this will be a multivariate time series. In the panel and prepanel functions, these are the data coordinates.
data	Not used (at least, not used by xyplot.ts).
...	further arguments. Arguments to xyplot as well as to the default panel function panel.horizonplot can be supplied directly to horizonplot. In typical usage, the method of xyplot called will be xyplot.ts.
nbands	Integer giving the number of discrete color bands used (for both negative and positive deviations from the origin).
horizonscale	the scale of each color segment. There are 3 positive segments and 3 negative segments. If this is a given as a number then all panels will have comparable distances, though not necessarily the same actual values (similar in concept to scales$relation = "sliced"). If NA, as it is by default, then the scale is chosen in each panel to cover the range of the data (unless overridden by ylim); see Details.
origin	the baseline y value for the first (positive) segment (i.e. the value at which red changes to blue). This can be a number, which is then fixed across all panels, or it can be a function, which is evaluated with the y values in each panel. The default is the first non-missing y value in each panel. See the Details section.
colorkey, legend	if colorkey = TRUE a suitable color scale bar is constructed using the values of origin and horizonscale. Further options can be passed to colorkey in list form, as with levelplot.
panel	function to render the graphic given the data. This is the function that actually implements the display.
prepanel	function determining range of the data rectangle from data to be used in a panel.
col.regions	color scale, with at least 6 colors. This should be a divergent color scale (typically with white as the central color).
strip, strip.left	by default strips are only drawn on the left, to save space.
par.strip.text	graphical parameters for the strip text; see xyplot. One notable argument here is lines, allowing multi-line text.
colorkey.digits	digits for rounding values in colorkey labels.
layout	Numeric vector of length 2 (or 3) specifying number of columns and rows (and pages) in the plot. The default is to have one column and as many rows as there are panels.
default.scales	sets default values of scales; leave this alone, pass scales instead.
groups	not applicable to this type of plot.
border	border color for the filled polygons, defaults to no border.

Details

This function draws time series as filled areas, with modifications to effectively visualise many time series in parallel. Data that would be drawn off the top of each panel is redrawn from the bottom of the panel in a darker color. Values below the origin are inverted and drawn in the opposite color. There are up to three shades (typically in blue) for data above the baseline and up to three shades (typically in red) for data below the baseline. See the article referenced below for an introduction to Horizon plots.

There are three different cases of using this function:

1. `horizonscale` unspecified (default case): then each panel will have different scales, and the colors represent deviations from the origin up to the maximum deviation from the origin in that panel. If `origin` is specified then that will be constant across panels; otherwise it defaults to the initial value.
2. `horizonscale` specified but `origin` unspecified: the origin defaults to the initial value in each panel, and colors represent deviations from it in steps of `horizonscale` (up to 3 steps each way).
3. both `horizonscale` and `origin` specified: each panel will have the same scales, and colors represent fixed ranges of values.

In each of these cases the `colorkey` is labelled slightly differently (see examples).

Value

An object of class `"trellis"`. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Warning

Note that the y scale in each panel defines the actual origin and scale used. The `origin` and `horizonscale` arguments are only used in the `prepanel` function to choose an appropriate y scale. The `ylim` argument therefore over-rides `origin` and `horizonscale`. This also implies that choices of `scales$y$relation` other than `"free"` may have unexpected effects, particularly `"sliced"`, as these change the y limits from those requested by the prepanel function.

Author(s)

Felix Andrews <felix@nfrac.org>

References

Stephen Few (2008). Time on the Horizon. *Visual Business Intelligence Newsletter*, June/July 2008
http://www.perceptualedge.com/articles/visual_business_intelligence/time_on_the_horizon.pdf

See Also

`Lattice`, `xyplot.ts`, `panel.xyarea`

Examples

```
## generate a random time series object with 12 columns
set.seed(1)
dat <- ts(matrix(cumsum(rnorm(200 * 12)), ncol = 12))
colnames(dat) <- paste("series", LETTERS[1:12])

## show simple line plot first, for reference.
xyplot(dat, scales = list(y = "same"))

## these layers show scale and origin in each panel...
infolayers <-
  layer(panel.scaleArrow(x = 0.99, digits = 1, col = "grey",
                         srt = 90, cex = 0.7)) +
  layer(lim <- current.panel.limits(),
    panel.text(lim$x[1], lim$y[1], round(lim$y[1],1), font = 2,
        cex = 0.7, adj = c(-0.5,-0.5), col = "#9FC8DC"))

## Case 1: each panel has a different origin and scale:
## ('origin' default is the first data value in each series).
horizonplot(dat, layout = c(1,12), colorkey = TRUE) +
  infolayers

## Case 2: fixed scale but different origin (baseline):
## (similar in concept to scales = "sliced")
horizonplot(dat, layout = c(1,12), horizonscale = 10, colorkey = TRUE) +
  infolayers

## Case 3: fixed scale and constant origin (all same scales):
horizonplot(dat, layout = c(1,12), origin = 0, horizonscale = 10, colorkey = TRUE) +
  infolayers

## same effect using ylim (but colorkey does not know limits):
horizonplot(dat, layout = c(1,12), ylim = c(0, 10), colorkey = TRUE) +
  infolayers

## same scales with full coverage of color scale:
horizonplot(dat, layout = c(1,12), origin = 0,
            scales = list(y = list(relation = "same")),
            colorkey = TRUE, colorkey.digits = 1) +
  infolayers


## use ylab rather than strip.left, for readability.
## also shade any times with missing data values.
horizonplot(dat, horizonscale = 10, colorkey = TRUE,
            layout = c(1,12), strip.left = FALSE,
            ylab = list(rev(colnames(dat)), rot = 0, cex = 0.7)) +
  layer_(panel.fill(col = "gray90"), panel.xblocks(..., col = "white"))


## illustration of the cut points used in the following plot
xyplot(EuStockMarkets, scales = list(y = "same"),
```

```
  panel = function(x, y, ...) {
    col <-
    c("#B41414","#E03231","#F7A99C","#9FC8DC","#468CC8","#0165B3")
    for (i in c(-3:-1, 2:0)) {
      if (i >= 0)
        yi <- pmax(4000, pmin(y, 4000 + 1000 * (i+1)))
      if (i < 0)
        yi <- pmin(4000, pmax(y, 4000 + 1000 * i))
      panel.xyarea(x, yi, origin = 4000,
        col = col[i+4], border = NA)
    }
    panel.lines(x, y)
    panel.abline(h = 4000, lty = 2)
  })

## compare with previous plot
horizonplot(EuStockMarkets, colorkey = TRUE,
            origin = 4000, horizonscale = 1000) +
  infolayers

## a cut-and-stack plot; use constant y scales!
horizonplot(sunspots, cut = list(n = 23, overlap = 0),
  scales = list(draw = FALSE, y = list(relation = "same")),
  origin = 100, colorkey = TRUE,
  strip.left = FALSE, layout = c(1,23)) +
layer(grid::grid.text(round(x[1]), x = 0, just = "left"))
```

`layer` *Add layers to a lattice plot, optionally using a new data source*

Description

A mechanism to add new layers to a trellis object, optionally using a new data source. This is an alternative to modifying the panel function. Note the non-standard evaluation in `layer()`.

Usage

```
layer(..., data, magicdots, exclude,
      packets, rows, columns, groups,
      style, force, theme, under, superpose)

layer_(...)
glayer(...)
glayer_(...)

## S3 method for class 'trellis'
object + lay

drawLayer(lay, panelArgs = trellis.panelArgs())
```

```
flattenPanel(object)
```

Arguments

`...`	expressions as they would appear in a panel function. These can refer to the panel function arguments (such as `x`, `y` and `subscripts`), and also to any named objects passed in through the `data` argument. The calls can also include the special argument "`...`"; in the default case of `magicdots = TRUE`, only those arguments which are not already named in a call are passed on through "`...`". Otherwise, "`...`" simply represents all panel function arguments. See Details, below.
`data`	optional. A named `list` containing objects needed when evaluating (drawing) the layer.
`magicdots, exclude`	if `magicdots = TRUE`, the default, any reference to "`...`" in the layer expressions will only pass on those arguments from the panel function which are not named in the call (thus avoiding duplicate argument errors). If the first argument in a call is not named, it is assumed to be named `"x"`, and if the second argument is not named it is assumed to be named `"y"`. Furthermore, any argument names given in `exclude` will not be passed on through "`...`".
`packets, rows, columns, groups`	restricts the layer to draw only in specified packets (which refer to individual panels, but are independent of their layout), or rows or columns of the trellis layout (`trellis.currentLayout`). For group layers (using `glayer` or `superpose = TRUE`), the groups can be restricted also, by specifying group numbers (or group values, as character strings). Negative values exclude the given items.
`style`	style index of the layer, used only to set lattice graphical parameters (same effect as in grouped displays). Note that this will use the theme settings in effect in the existing plot, which may or may not be what is desired. It may be necessary to use `force = TRUE` to escape from the plot's settings and use the current theme.
`force`	`force = TRUE` is just a shorthand for `theme = trellis.par.get()`, which is useful for over-riding the theme settings in effect in an existing plot. For instance, if the original plot specified `par.settings = simpleTheme(col = "red")` then the theme settings in effect will be entirely red. Use `force = TRUE` to reset the current theme for this layer, or use `theme` directly.
`theme`	a style specification to be passed to `trellis.par.set` which has effect only while drawing the layer. One can pass a whole theme specification list, such as `theme = custom.theme()`, or a more specific list, such as `theme = simpleTheme(col = "red")`.
`under`	whether the layer should be drawn before the existing panel function. This defaults to `TRUE` in the convenience functions `layer_()` and `glayer_()`.
`superpose`	if `TRUE`, the layer will be drawn once for each level of any `groups` in the plot, using `panel.superpose`. This defaults to `TRUE` in the convenience functions `glayer()` and `glayer_()`.
`object`	a trellis object.

lay	a layer object.
panelArgs	list of arguments to the panel function.

Details

The layer mechanism is a method for augmenting a panel function. It allows expressions to be added to the panel function without knowing what the original panel function was. In this way it can be useful for convenient augmentation of trellis plots.

Note that the evaluation used in layer is non-standard, and can be confusing at first: you typically refer to variables as if inside the panel function (x, y, etc); you can usually refer to objects which exist in the global environment (workspace), but it is safer to pass them in by name in the data argument to layer. (And this should not to be confused with the data argument to the original xyplot.)

A simple example is adding a reference line to each panel: layer(panel.refline(h = 0)). Note that the expressions are quoted, so if you have local variables they will need to be either accessible globally, or passed in via the data argument. For example:

```
layer(panel.refline(h = myVal)) ## if myVal is global
```

```
layer(panel.refline(h = h),data = list(h = myVal))
```

Another non-standard aspect is that the special argument "..." will, by default, only pass through those argument not already named. For example, this will over-ride the x argument and pass on the remaining arguments:

```
layer(panel.xyplot(x = jitter(x),...))
```

The first un-named argument is assumed to be "x", so that is the same as

```
layer(panel.xyplot(jitter(x),...))
```

The layer mechanism should probably still be considered experimental.

drawLayer() actually draws the given layer object, applying the panel specification, style settings and so on. It should only be called while a panel is in focus.

The flattenPanel function will construct a human-readable function incorporating code from all layers (and the original panel function). Note that this does not return a usable function, as it lacks the correct argument list and ignores any extra data sources that layers might use. It is intended be edited manually.

Value

a layer object is defined as a list of expression objects, each of which may have a set of attributes. The result of "adding" a layer to a trellis object (+.trellis) is the updated trellis object.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

update.trellis, as.layer for overlaying entire plots

Examples

```
foo <- xyplot(ozone ~ wind, environmental)
foo

## overlay reference lines
foo <- foo + layer(panel.abline(h = 0)) +
             layer(panel.lmline(x, y, lty = 2))

## underlay a flat color
foo <- foo + layer(panel.fill(grey(.95)), under = TRUE)
foo

## layers can access the panel function arguments
foo <- foo + layer({ ok <- (y>100);
            panel.text(x[ok], y[ok], y[ok], pos = 1) })
foo

## over-ride arguments by name
foo <- foo +
  layer(panel.xyplot(y = ave(y, x, FUN = max), type = "a", ...))
foo

## see a sketch of the complete panel function
flattenPanel(foo)


## group layers, drawn for each group in each panel
dotplot(VADeaths, type = "o") +
  glayer(ltext(x[5], y[5], group.value, srt = 40))

## a quick way to print out the panel.groups arguments:
dotplot(VADeaths, type = "o") + glayer(str(list(...)))


## layers with superposed styles
xyplot(ozone ~ wind | equal.count(temperature, 2),
       data = environmental) +
   layer(panel.loess(x, y, span = 0.5), style = 1) +
   layer(panel.loess(x, y, span = 1.0), style = 2) +
   layer(panel.key(c("span = 0.5", "span = 1.0"), corner = c(1,.98),
                   lines = TRUE, points = FALSE), packets = 1)

## note that styles come from the settings in effect in the plot,
## which is not always what you want:
xyplot(1:10 ~ 1:10, type = "b", par.settings = simpleTheme(col = "red")) +
  layer(panel.lines(x = jitter(x, 2), ...)) +  ## drawn in red
  layer(panel.lines(x = jitter(x, 2), ...), force = TRUE)  ## reset theme


## using other variables from the original `data` object
## NOTE: need subscripts = TRUE in original call!
zoip <- xyplot(wind ~ temperature | equal.count(radiation, 2),
```

```
   data = environmental, subscripts = TRUE)
zoip + layer(panel.points(..., pch = 19,
             col = grey(1 - ozone[subscripts] / max(ozone))),
       data = environmental)


## restrict drawing to specified panels
barchart(yield ~ variety | site, data = barley,
         groups = year, layout = c(1,6), as.table = TRUE,
         scales = list(x = list(rot = 45))) +
layer(ltext(tapply(y, x, max), lab = abbreviate(levels(x)),
      pos = 3), rows = 1)


## example of a new data source
qua <- xyplot(lat ~ long | cut(depth, 2), quakes,
    aspect = "iso", pch = ".", cex = 2)
qua
## add layer showing distance from Auckland
newdat <- with(quakes, expand.grid(
            gridlat = seq(min(lat), max(lat), length = 60),
            gridlon = seq(min(long), max(long), length = 60)))
newdat$dist <- with(newdat, sqrt((gridlat - -36.87)^2 +
                                 (gridlon - 174.75)^2))
qua + layer_(panel.contourplot(x = gridlon, y = gridlat, z = dist,
   contour = TRUE, subscripts = TRUE), data = newdat)
```

---

mapplot *Trellis displays on Maps a.k.a. Choropleth maps*

Description

Produces Trellis displays of numeric (and eventually categorical) data on a map. This is largely meant as a demonstration, and users looking for serious map drawing capabilities should look elsewhere (see below).

Usage

```
mapplot(x, data, ...)

## S3 method for class 'formula'
mapplot(x, data, map, outer = TRUE,
        prepanel = prepanel.mapplot,
        panel = panel.mapplot,
        aspect = "iso",
        legend = NULL,
        breaks, cuts = 30,
        colramp = colorRampPalette(hcl.colors(n = 11, palette = "Spectral")),
```

```
          colorkey = TRUE,
          ...)


prepanel.mapplot(x, y, map, ...)
panel.mapplot(x, y, map, breaks, colramp, exact = FALSE, lwd = 0.5, ...)
```

Arguments

x, y	For mapplot, an object on which method dispatch is carried out. For the formula method, a formula of the form y ~ x, with additional conditioning variables as desired. The extended form of conditioning using y ~ x1 + x2 etc. is also allowed. The formula might be interpreted as in a dot plot, except that y is taken to be the names of geographical units in map. Suitable subsets (packets) of x and y are passed to the prepanel and panel functions.
data	A data source where names in the formula are evaluated
map	An object of class "map" (package maps), containing boundary information. The names of the geographical units must match the y variable in the formula. The remaining arguments are standard lattice arguments, relvant here mostly because they have different defaults than usual:
outer	logical; how variables separated by + in the formula are interpreted. It is not advisable to change the default.
prepanel, panel	the prepanel and panel functions
aspect	aspect ratio
breaks, cuts, colramp	controls conversion of numeric x values to a false color. colramp should be a function that produces colors (such as cm.colors). If it is NULL, colors are taken from trellis.par.get("regions").
exact	the default exact = FALSE allows the given y values to match sub-regions of map, i.e. region names with a qualifier following ":", like "michigan:north","michigan:south". These will both match a y value of "Michigan".
legend, colorkey	controls legends; usually just a color key giving the association between numeric values of x and color.
lwd	line width
...	Further arguments passed on to the underlying engine. See xyplot for details.

Value

An object of class "trellis".

Note

This function is meant to demonstrate how maps can be incorporated in a Trellis display. Users seriously interested in geographical data should consider using software written by people who know what they are doing.

Author(s)

Deepayan Sarkar

References

http://en.wikipedia.org/wiki/Choropleth_map

See Also

Lattice

Examples

```
library(maps)
library(mapproj)

## Note: Alaska, Hawaii and others are not included in county map;
## this generates warnings with both USCancerRates and ancestry.

data(USCancerRates)

suppressWarnings(print(
mapplot(rownames(USCancerRates) ~ log(rate.male) + log(rate.female),
        data = USCancerRates,
        map = map("county", plot = FALSE, fill = TRUE,
                  projection = "mercator"))
))

suppressWarnings(print(
mapplot(rownames(USCancerRates) ~ log(rate.male) + log(rate.female),
        data = USCancerRates,
        map = map("county", plot = FALSE, fill = TRUE,
                  projection = "tetra"),
        scales = list(draw = FALSE))
))

data(ancestry)

county.map <-
    map('county', plot = FALSE, fill = TRUE,
        projection = "azequalarea")

## set a sequential color palette as current theme, and use it
opar <- trellis.par.get()
trellis.par.set(custom.theme(region = rev(hcl.colors(9, "Purp")),
```

```
                              alpha.line = 0.5))
suppressWarnings(print(
mapplot(county ~ log10(population), ancestry, map = county.map,
    colramp = NULL, border = "transparent")
))
trellis.par.set(opar)

## Not run:

## this may take a while (should get better area records)

county.areas <-
    area.map(county.map, regions = county.map$names, sqmi = FALSE)

ancestry$density <-
    with(ancestry, population / county.areas[as.character(county)])

mapplot(county ~ log(density), ancestry,
        map = county.map, border = NA,
        colramp = colorRampPalette(c("white", "black")))


## End(Not run)
```

`marginal.plot` *Display marginal distributions*

Description

Display marginal distributions of several variables, which may be numeric and/or categorical, on one plot.

Usage

```
marginal.plot(x,
              data = NULL,
              groups = NULL,
              reorder = !is.table(x),
              plot.points = FALSE,
              ref = TRUE, cut = 0,
              origin = 0,
              xlab = NULL, ylab = NULL,
              type = c("p", if (is.null(groups)) "h"),
              ...,
              subset = TRUE,
              as.table = TRUE,
              subscripts = TRUE,
              default.scales = list(
```

```
    relation = "free",
    abbreviate = TRUE, minlength = 5,
    rot = 30, cex = 0.75, tick.number = 3,
    y = list(draw = FALSE)),
  layout = NULL,
  lattice.options = list(
    layout.heights = list(
      axis.xlab.padding = list(x = 0),
      xlab.key.padding = list(x = 0))))
```

Arguments

x	a data frame or table, or a formula of which the first term is a data frame or table. Otherwise coerced with as.data.frame.
data	an optional data source in which groups and subset may be be evaluated.
groups	term, to be evaluated in data, that is used as a grouping variable.
reorder	whether to reorder factor variables by frequency.
subset	data subset expression, evaluated in data.
plot.points, ref, cut	passed to panel.densityplot.
origin, type	passed to panel.dotplot.
xlab, ylab, as.table, subscripts	see xyplot.
default.scales, layout, lattice.options	see xyplot.
...	passed to panel.densityplot and/or panel.dotplot.

Details

In the case of mixed numeric and categorical variables, the trellis objects from dotplot() and densityplot() are merged.

Value

a trellis object.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

panel.dotplot, panel.densityplot

Examples

```
enviro <- environmental
## make an ordered factor (so it will not be reordered)
enviro$smell <- cut(enviro$ozone, breaks = c(0, 30, 50, Inf),
    labels = c("ok", "hmmm", "yuck"), ordered = TRUE)
marginal.plot(enviro)

## using groups
enviro$is.windy <- factor(enviro$wind > 10,
    levels = c(TRUE, FALSE), labels = c("windy", "calm"))
marginal.plot(enviro[,1:5], data = enviro, groups = is.windy,
    auto.key = list(lines = TRUE))

## support for tables
marginal.plot(Titanic)
## table with groups
marginal.plot(~ Titanic, data = Titanic, groups = Survived,
    type = "b", auto.key = list(title = "Survived?"))
```

---

panel.2dsmoother *Plot a smooth approximation of z over x and y.*

Description

Plot a smooth approximation, using loess by default, of one variable (z) against two others (x and y).

This panel function should be used with a levelplot.

Usage

```
panel.2dsmoother(x, y, z, subscripts = TRUE,
    form = z ~ x * y, method = "loess", ...,
    args = list(), n = 100)
```

Arguments

x, y, z	data points. If these are missing, they will be looked for in the environment of form. So in many cases you can skip these if passing form. In fact, for convenience, the formula can be passed as the first argument (i.e. x).
form, method	the smoothing model is constructed (approximately) as method(form,data = list(x=x,y=y,z=z),{args}). See the Examples section for common choices.
subscripts	data indices for the current packet, as passed in by levelplot.
...	further arguments passed on to panel.levelplot.
args	a list of further arguments to the model function (method).
n	number of equi-spaced points along each of x and y on which to evaluate the smooth function.

Details

This should work with any model function that takes a formula argument, and has a `predict` method argument.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

loess, panel.smoother

Examples

```
set.seed(1)
xyz <- data.frame(x = rnorm(100), y = rnorm(100))
xyz$z <- with(xyz, x * y + rnorm(100, sd = 1))

levelplot(z ~ x * y, xyz, panel = panel.2dsmoother)

## showing data points on the same color scale
levelplot(z ~ x * y, xyz,
          panel = panel.levelplot.points, cex = 1.2) +
  layer_(panel.2dsmoother(..., n = 200))

## simple linear regression model
levelplot(z ~ x * y, xyz,
          panel = panel.levelplot.points) +
  layer_(panel.2dsmoother(..., method = "lm"))

## GAM smoother with smoothness by cross validation
if (require("mgcv"))
  levelplot(z ~ x * y, xyz, panel = panel.2dsmoother,
            form = z ~ s(x, y), method = "gam")
```

---

panel.3dmisc *Miscellanous panel utilities for three dimensional Trellis Displays*

Description

Miscellanous panel functions for use with three dimensional Lattice functions such as cloud and wireframe

Usage

```
panel.3dbars(x, y, z,
             rot.mat = diag(4), distance,
             xbase = 1, ybase = 1,
```

```
              xlim, xlim.scaled,
              ylim, ylim.scaled,
              zlim, zlim.scaled,
              zero.scaled,
              col = "black",
              lty = 1, lwd = 1,
              alpha,
              ...,
              col.facet = "white",
              alpha.facet = 1)


panel.3dpolygon(x, y, z, rot.mat = diag(4), distance,
                xlim.scaled,
                ylim.scaled,
                zlim.scaled,
                zero.scaled,
                col = "white",
                border = "black",
                font, fontface,
                ...)

panel.3dtext(x, y, z, labels = seq_along(x),
             rot.mat = diag(4), distance, ...)
```

Arguments

`x, y, z`	data to be plotted
`rot.mat, distance`	arguments controlling projection
`labels`	character or expression vectors to be uses as labels
`xlim, ylim, zlim`	limits in the original scale
`xlim.scaled, ylim.scaled, zlim.scaled`	limits after scaling
`zero.scaled`	the value of z = 0 after scaling
`xbase, ybase`	length of the sides of the bars (which are always centered on the `x` and `y` values). Can not be vectorized.
`col, lty, lwd, alpha, border`	Graphical parameters for the border lines. These can be vectors, in which case each component will be associated with one bar in `panel.3dbars`.
`font, fontface`	unused graphical parameters, present in the argument list only so that they can be captured and ignored
`col.facet, alpha.facet`	Graphical parameters for surfaces of the bars . These can be vectors, in which case each component will be associated with one bar.

... extra arguments, passed on as appropriate.

Details

panel.3dbars and panel.3dpolygon are both suitable for use as (components of) the panel.3d.cloud argument of panel.cloud. The first one produces three dimensional bars, and the second one draws three dimensional polygons.

Author(s)

Deepayan Sarkar <deepayan.sarkar@gmail.com>

See Also

cloud, panel.cloud

Examples

```
library(lattice)

cloud(VADeaths, panel.3d.cloud = panel.3dbars,
      col.facet = "grey", xbase = 0.4, ybase = 0.4,
      screen = list(z = 40, x = -30))

cloud(VADeaths, panel.3d.cloud = panel.3dbars,
      xbase = 0.4, ybase = 0.4, zlim = c(0, max(VADeaths)),
      scales = list(arrows = FALSE, just = "right"), xlab = NULL, ylab = NULL,
      col.facet = level.colors(VADeaths, at = do.breaks(range(VADeaths), 20),
                               col.regions = terrain.colors,
                               colors = TRUE),
      colorkey = list(col = terrain.colors, at = do.breaks(range(VADeaths), 20)),
      screen = list(z = 40, x = -30))


cloud(as.table(prop.table(Titanic, margin = 1:3)[,,,2]),
      type = c("p", "h"),
      zlab = "Proportion\nSurvived",
      panel.3d.cloud = panel.3dbars,
      xbase = 0.4, ybase = 0.4,
      aspect = c(1, 0.3),
      scales = list(distance = 2),
      panel.aspect = 0.5)
```

panel.ellipse *Lattice panel function to fit and draw a confidence ellipsoid from bivariate data.*

Description

A lattice panel function that computes and draws a confidence ellipsoid from bivariate data, possibly grouped by a third variable.

Usage

```
panel.ellipse(x, y, groups = NULL,
              level = 0.68, segments = 50, robust = FALSE,
              center.pch = 3, center.cex = 2, ...,
              type, pch, cex)
```

Arguments

x, y	Numeric vectors of same length giving the bivariate data. Non-numeric variables will be coerced to be numeric.
groups	Optional grouping variable.
level	Confidence level for the ellipse.
segments	Number of segments used to approximate the ellipse.
robust	Logical indicating whether a robust method should be used. If TRUE, the confidence ellipse is based on a bivariate t-distribution using the cov.trob function in the **MASS** package.
center.pch	Plotting character for the center (fitted mean). If NULL, the center will not be shown on the plot.
center.cex	Character expansion (size) multipler for the symbol indicating the center.
...	Further arguments, typically graphical parameters. Passed on to panel.xyplot.
type, pch, cex	Parameters that are ignored; these are present only to make sure they are not inadvertently passed on to panel.xyplot.

Author(s)

Deepayan Sarkar, extending code contributed by Michael Friendly.

Examples

```
xyplot(Sepal.Length ~ Petal.Length, groups=Species,
       data = iris, scales = "free",
       par.settings = list(superpose.symbol = list(pch=c(15:17)),
                           superpose.line = list(lwd=2, lty=1:3)),
       panel = function(x, y, ...) {
           panel.xyplot(x, y, ...)
```

```
            panel.ellipse(x, y, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)))

## Without groups
xyplot(Sepal.Length ~ Petal.Length,
        data = iris, scales = "free",
        par.settings = list(plot.symbol = list(cex = 1.1, pch=16)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.ellipse(x, y, lwd = 2, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)))


## With conditioning
xyplot(Sepal.Length ~ Petal.Length | Species,
        data = iris, scales = "free",
        par.settings = list(plot.symbol = list(cex = 1.1, pch=16)),
        layout=c(2,2),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.ellipse(x, y, lwd = 2, ...)
        },
        auto.key = list(x = .6, y = .8, corner = c(0, 0)))


## Compare classical with robust
xyplot(Sepal.Length ~ Petal.Length | Species,
        data = iris, scales = "free",
        par.settings = list(plot.symbol = list(cex = 1.1, pch=16)),
        layout=c(2,2),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.ellipse(x, y, lwd = 2, ...)
            panel.ellipse(x, y, lwd = 2, col="red", robust=TRUE,  ...)
        })
```

`panel.key` *Draw a simple key inside a panel of a lattice plot.*

Description

Draw a `simpleKey` inside a panel of a lattice plot.

Usage

```
panel.key(text, ..., corner = c(0, 1), x = corner[1], y = corner[2])
```

Arguments

text, ...	entries in the simpleKey.
corner, x, y	defines the position of the key within the panel viewport. These are given in normalised coordinates between 0 and 1. The two elements of corner specify the x and y positions respectively.

See Also

simpleKey, draw.key, xyplot

Examples

```
xyplot(ozone ~ wind | equal.count(temperature, 2),
       data = environmental) +
   layer(panel.loess(x, y, span = 0.5), style = 1) +
   layer(panel.loess(x, y, span = 1.0), style = 2) +
   layer(panel.key(c("span = 0.5", "span = 1.0"), corner = c(1,.98),
                   lines = TRUE, points = FALSE), packets = 1)
```

---

panel.lmlineq *Draw a line with a label, by default its equation*

Description

This is an extension of the panel functions panel.abline and panel.lmline to also draw a label on the line. The default label is the line equation, and optionally the R squared value of its fit to the data points.

Usage

```
panel.ablineq(a = NULL, b = 0,
              h = NULL, v = NULL,
              reg = NULL, coef = NULL,
              pos = if (rotate) 1 else NULL,
              offset = 0.5, adj = NULL,
              at = 0.5, x, y,
              rotate = FALSE, srt = 0,
              label = NULL,
              varNames = alist(y = y, x = x),
              varStyle = "italic",
              fontfamily = "serif",
              digits = 3,
              r.squared = FALSE, sep = ", ", sep.end = "",
              col, col.text, col.line,
              ..., reference = FALSE)

panel.lmlineq(x, y, ...)
```

Arguments

`a, b, h, v, reg, coef`
: specification of the line. The simplest usage is to give `a` and `b` to describe the line $y = a + b\,x$. Horizontal or vertical lines can be specified as arguments `h` or `v`, respectively. The first argument (`a`) can also be a model object produced by `lm`. See `panel.abline` for more details.

`pos, offset`
: passed on to `panel.text`. For `pos`: 1 = below, 2 = left, 3 = above, 4 = right, and the `offset` (in character widths) is applied.

`adj`
: passed on to `panel.text`. c(0,0) = above right, c(1,0) = above left, c(0,1) = below right, c(1,1) = below left; offset does not apply when using `adj`.

`fontfamily`
: passed on to `panel.text`.

`at`
: position of the equation as a fractional distance along the line. This should be in the range 0 to 1. When a vertical line is drawn, this gives the vertical position of the equation.

`x, y`
: position of the equation in native units. If given, this over-rides `at`. For `panel.lmlineq` this is the data, passed on as `lm(y ~ x)`.

`rotate, srt`
: set `rotate = TRUE` to align the equation with the line. This will over-ride `srt`, which otherwise gives the rotation angle. Note that the calculated angle depends on the current device size; this will be wrong if you change the device aspect ratio after plotting.

`label`
: the text to draw along with the line. If specified, this will be used instead of an equation.

`varNames`
: names to display for `x` and/or `y`. This should be a list like `list(y = "Q", x = "X")` or, for mathematical symbols, `alist(y = (alpha + beta), x = sqrt(x[t]))`.

`varStyle`
: the name of a `plotmath` function to wrap around the equation expression, or `NULL`. E.g. `"bolditalic"`, `"displaystyle"`.

`digits`
: number of decimal places to show for coefficients in equation.

`r.squared`
: the R^2 statistic to display along with the equation of a line. This can be given directly as a number, or `TRUE`, in which case the function expects a model object (typically `lm`) and extracts the R^2 statistic from it.

`sep, sep.end`
: The R^2 (`r.squared`) value is separated from the equation by the string `sep`, and also `sep.end` is added to the end. For example: `panel.ablineq(lm(y ~ x), r.squared = TRUE, sep = " (", sep.end = ")")`.

`..., col, col.text, col.line`
: passed on to `panel.abline` and `panel.text`. Note that `col` applies to both text and line; `col.text` applies to the equation only, and `col.line` applies to line only.

`reference`
: whether to draw the line in a "reference line" style, like that used for grid lines.

Details

The equation is constructed as an expression using `plotmath`.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

panel.abline, panel.text, lm, plotmath

Examples

```
set.seed(0)
xsim <- rnorm(50, mean = 3)
ysim <- (0 + 2 * xsim) * (1 + rnorm(50, sd = 0.3))

## basic use as a panel function
xyplot(ysim ~ xsim, panel = function(x, y, ...) {
  panel.xyplot(x, y, ...)
  panel.ablineq(a = 0, b = 2, adj = c(0,1))
  panel.lmlineq(x, y, adj = c(1,0), lty = 2,
                col.line = "grey", digits = 1)
})

## using layers:
xyplot(ysim^2 ~ xsim) +
  layer(panel.ablineq(lm(y ~ x, subset = x <= 3),
    varNames = alist(y = y^2, x = x[x <= 3]), pos = 4))

## rotated equation (depends on device aspect at plotting time)
xyplot(ysim ~ xsim) +
  layer(panel.ablineq(lm(y ~ x), rotate = TRUE, at = 0.8))

## horizontal and vertical lines
xyplot(ysim ~ xsim) +
  layer(panel.ablineq(v = 3, pos = 4, at = 0.1, lty = 2,
                      label = "3.0 (critical value)")) +
  layer(panel.ablineq(h = mean(ysim), pos = 3, at = 0.15, lty = 2,
                      varNames = alist(y = plain(mean)(y))))

## using layer styles, r.squared
xyplot(ysim ~ xsim) +
  layer(panel.ablineq(lm(y ~ x), r.sq = TRUE,
                      at = 0.4, adj=0:1), style = 1) +
  layer(panel.ablineq(lm(y ~ x + 0), r.sq = TRUE,
                      at = 0.6, adj=0:1), style = 2)

## alternative placement of equations
xyplot(ysim ~ xsim) +
  layer(panel.ablineq(lm(y ~ x), r.sq = TRUE, rot = TRUE,
                      at = 0.8, pos = 3), style = 1) +
  layer(panel.ablineq(lm(y ~ x + 0), r.sq = TRUE, rot = TRUE,
                      at = 0.8, pos = 1), style = 2)

update(trellis.last.object(),
```

```
auto.key = list(text = c("intercept", "no intercept"),
                points = FALSE, lines = TRUE))
```

`panel.qqmath.tails` *Approximate distribution in qqmath but keep points on tails.*

Description

Panel function for `qqmath` to reduce the number of points plotted by sampling along the specified distribution. The usual method for such sampling is to use the `f.value` argument to `panel.qqmath`. However, this panel function differs in two ways: (1) a specified number of data points are retained (not interpolated) on each tail of the distribution. (2) the sampling is evenly spaced along the specified distribution automatically (whereas `f.value = ppoints(100)` is evenly spaced along the uniform distribution only).

This function is deprecated as of **lattice** *0.18-4 (available for R 2.11.0). Use the* `tails.n` *argument of* `panel.qqmath` *instead.*

Usage

```
panel.qqmath.tails(x, f.value = NULL, distribution = qnorm,
                   groups = NULL, ..., approx.n = 100, tails.n = 10)
```

Arguments

`x, f.value, distribution, groups`	see `panel.qqmath`.
`...`	further arguments passed on to `panel.xyplot`.
`approx.n`	number of points to use in approximating the distribution. Points will be equally spaced in the distribution space.
`tails.n`	number of points to retain (untouched) at both the high and low tails.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`panel.qqmath` which should be used instead (as of **lattice** 0.18-4).

Examples

```
## see ?panel.qqmath
```

panel.quantile *Plot a quantile regression line with standard error bounds.*

Description

Plot a quantile regression line with standard error bounds, using the **quantreg** package. This is based on the stat_quantile function from **ggplot2**.

Usage

```
panel.quantile(x, y, form = y ~ x, method = "rq", ...,
    tau = 0.5, ci = FALSE, ci.type = "default", level = 0.95,
    n = 100, col = plot.line$col, col.se = col,
    lty = plot.line$lty, lwd = plot.line$lwd,
    alpha = plot.line$alpha, alpha.se = 0.25, border = NA,
    superpose = FALSE,
    ## ignored: ##
    subscripts, group.number, group.value,
    type, col.line, col.symbol, fill,
    pch, cex, font, fontface, fontfamily)
```

Arguments

x, y	data points. If these are missing, they will be looked for in the environment of form. So in many cases you can skip these if passing form. In fact, for convenience, the formula can be passed as the first argument (i.e. x).
form, method	Formula and the name of a modeling function (as a character string). The model is constructed (approximately) as method(form,tau = tau,data = list(x=x,y=y),...). Currently, method is expected to be either "rq" or "rqss", and for these the corresponding functions in the **quantreg** package is used. In principle, any other function that supports the same interface can also be specified by name.
...	further arguments passed on to the model function (method), typically rq.
tau	*p* values for the quantiles to estimate. Note: only one value for tau can be specified if estimating confidence intervals with ci.
ci, ci.type, level	estimate a confidence interval at level level using the method ci.type; see predict.rq.
n	number of equi-spaced points on which to evaluate the function.
col, col.se, lty, lwd, alpha, alpha.se, border	graphical parameters. col and alpha apply to the line(s), while col.se and alpha.se apply to the shaded ci region.
superpose	if TRUE, plot each quantile line (tau) in a different style (using trellis.par.get("superpose.line")).
subscripts, group.number, group.value, type, col.line, col.symbol, fill, pch, cex, font, fontface, fontfa	ignored.

Details

It is recommended to look at `vignette("rq",package="quantreg")`.

Author(s)

Felix Andrews <felix@nfrac.org>

Based on `stat_quantile` by Hadley Wickham.

See Also

`rq`, `panel.smoother`, `stat_quantile`

Examples

```
## library("quantreg")

set.seed(1)
xy <- data.frame(x = runif(100), y = rt(100, df = 5))
xyplot(y ~ x, xy) +
    layer(panel.quantile(x, y, tau = c(.95, .5, .05)))

if (require("splines")) {
    xyplot(y ~ x, xy) +
        layer(panel.quantile(y ~ ns(x, 3), tau = 0.9))

    xyplot(y ~ x, xy) +
        layer(panel.quantile(y ~ ns(x, 3), tau = 0.9, ci = TRUE))
}

xyplot(y ~ x, xy) +
    layer(panel.quantile(x, y, tau = c(.5, .9, .1), superpose = TRUE))

update(trellis.last.object(),
       auto.key = list(text = paste(c(50,90,10), "% quantile"),
                  points = FALSE, lines = TRUE))

xyplot(y ~ x, xy) +
    layer(panel.quantile(y ~ qss(x, lambda=1), method = "rqss"))
```

`panel.scaleArrow` *Draw a scale bar as an arrow, labelled with its length in plot units.*

Description

Draw a scale bar as an arrow, labelled with its length in plot units.

Usage

```
panel.scaleArrow(x = unit(0:1, "npc"), y = unit(0:1, "npc"),
                 default.units = "npc",
                 digits = 0, append = "", label = NULL,
                 angle = 30, length = 0.5, unit = "char",
                 type = "open", ends = "both",
                 ...,
                 col = add.line$col, fill = col, alpha = add.line$alpha,
                 lty = add.line$lty, lwd = add.line$lwd,
                 col.text = add.text$col, alpha.text = add.text$alpha)
```

Arguments

`x, y, default.units`	coordinates of the line ends as **grid** `unit`s or otherwise interpreted in `default.units`.
`digits`	number of decimal places to keep for the distance measure.
`append`	a string to append to the distance for the label.
`label`	label to place on the mid point of the scale, over-riding the default.
`angle, length, unit, type, ends`	specification of the arrow style; see `panel.arrows`.
`...`	further arguments passed to `panel.text`. You will need at least the `pos` or `adj` arguments.
`col, fill, alpha, lty, lwd`	graphical parameters relevant to the line.
`col.text, alpha.text`	graphical parameters relevant to the text label. Others like `cex` and `font` can be passed though `....`

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`panel.abline`, `grid.text`

Examples

```
xyplot(EuStockMarkets) +
  layer(panel.scaleArrow(x = 0.99, append = " units",
          col = "grey", srt = 90, cex = 0.8))
```

panel.segplot *Default prepanel and panel functions for segplot*

Description

Draws line segments or rectangles. Mainly intended to be used in conjunction with the segplot function.

Usage

```
prepanel.segplot(x, y, z, subscripts, horizontal = TRUE, ...)

panel.segplot(x, y, z, level = NULL, subscripts,
              at,
              draw.bands = is.factor(z),
              col, alpha,
              lty, lwd,
              border,
              col.symbol = col, alpha.symbol = alpha,
              col.regions = regions$col,
              band.height = 0.6,
              horizontal = TRUE,
              ...,
              segments.fun = panel.segments,
              centers = NULL,
              pch = 16)
```

Arguments

x, y, z	Vectors corresponding to x1, x2 and y respectively in the segplot formula. The names are different for compatibility with panel.levelplot. These are all the original vectors in data, not subsetted for particular panels.
level	optional vector controlling color of segments
centers	optional vector of 'centers' of the segments. If specified, points will be plotted at these y-locations.
pch	plotting character used for centers.
subscripts	integer subscript to be used as an indexing vector for x, y, z and level, giving the packet for the current panel.
horizontal	logical, whether the segments are to be drawn horizontally (the default) or vertically. This essentially swaps the role of the x- and y-axes in each panel.
at	values of level where color code changes
draw.bands	logical, whether to draw rectangles instead of lines
col, alpha, lty, lwd, border	Graphical parameters for the segment. Defaults to parameter settings for "plot.line" or "plot.polygon" for segments and rectangles respectively. col is overridden by col.regions if level is not null.

`col.symbol, alpha.symbol`
: Graphical parameters for the point if `centers` are plotted. Defaults to the corresponding parameters for the segment.

`col.regions`
: vector of colors as in `levelplot`

`band.height`
: height of rectangles (applicable if `draw.bands` is `TRUE`

`...`
: Other arguments, passed on to `panel.rect` (when `draw.bands=TRUE`), `segments.fun` (otherwise), `panel.points` (if `centers` is not `NULL`), etc. as appropriate.

`segments.fun`
: function used to plot segments when `draw.bands` is `FALSE`. The default is to use `panel.segments`, but `panel.arrows` is a useful alternative (arguments to `segments.fun` can be provided via the `...` argument, see example for `segplot`).

Value

For `prepanel.segplot` a list with components `xlim` and `ylim`.

Author(s)

Deepayan Sarkar <deepayan.sarkar@r-project.org>

See Also

`segplot`

---

panel.smoother *Plot a smoothing line with standard error bounds.*

Description

Plot a smoothing line with standard error bounds. This is based on the `stat_smooth` function from **ggplot2**.

Usage

```
panel.smoother(x, y, form = y ~ x, method = "loess", ...,
    se = TRUE, level = 0.95, n = 100,
    col = plot.line$col, col.se = col,
    lty = plot.line$lty, lwd = plot.line$lwd,
    alpha = plot.line$alpha, alpha.se = 0.25, border = NA,
    ## ignored: ##
    subscripts, group.number, group.value,
    type, col.line, col.symbol, fill,
    pch, cex, font, fontface, fontfamily)
```

Arguments

x, y	data points. If these are missing, they will be looked for in the environment of form. So in many cases you can skip these if passing form. In fact, for convenience, the formula can be passed as the first argument (i.e. x).
form, method	the smoothing model is constructed (approximately) as method(form,data = list(x=x,y=y),...). See the Examples section for common choices.
...	further arguments passed on to the model function (method).
se, level	estimate standard errors on the smoother, at the given level, and plot these as a band.
n	number of equi-spaced points on which to evaluate the smooth function.
col, col.se, lty, lwd, alpha, alpha.se, border	graphical parameters. col and alpha apply to the smoothing line, while col.se and alpha.se apply to the shaded se region.
subscripts, group.number, group.value, type, col.line, col.symbol, fill, pch, cex, font, fontface, fontfa	ignored.

Details

This should work with any model function that takes a formula argument, and has a predict method with a se argument.

Author(s)

Felix Andrews <felix@nfrac.org>

Based on stat_smooth by Hadley Wickham.

See Also

panel.loess, panel.quantile, stat_smooth

Examples

```
set.seed(1)
xy <- data.frame(x = runif(100),
                 y = rt(100, df = 5),
                 y2 = rt(100, df = 5) + 1)

xyplot(y ~ x, xy, panel = function(...) {
       panel.xyplot(...)
       panel.smoother(..., span = 0.9)
})

## per-group layers with glayer (pass `...` to get styles)
xyplot(y + y2 ~ x, xy) +
  glayer(panel.smoother(...))

## natural spline with 5 degrees of freedom
if (require("splines"))
```

```
  xyplot(y ~ x, xy) +
    layer(panel.smoother(y ~ ns(x,5), method = "lm"))

## thin plate regression spline with smoothness
## chosen by cross validation (see ?mgcv::gam)
if (require("mgcv"))
  xyplot(y ~ x, xy) +
    layer(panel.smoother(y ~ s(x), method = "gam"))

## simple linear regression with standard errors:
xyplot(y ~ x, xy) +
  layer(panel.smoother(x, y, method = "lm"), style = 2)
```

`panel.tskernel` *Calculate and plot smoothed time series.*

Description

Plot time series smoothed by discrete symmetric smoothing kernels. These kernels can be used to smooth time series objects. Options include moving averages, triangular filters, or approximately Gaussian filters.

Usage

```
panel.tskernel(x, y, ...,
    width = NROW(x) %/% 10 + 1, n = 300,
    c = 1, sides = 2, circular = FALSE,
    kern = kernel("daniell",
           rep(floor((width/sides) / sqrt(c)), c)))

simpleSmoothTs(x, ...)

## Default S3 method:
simpleSmoothTs(x, ...,
    width = NROW(x) %/% 10 + 1, n = NROW(x),
    c = 1, sides = 2, circular = FALSE,
    kern = kernel("daniell",
           rep(floor((width/sides)/sqrt(c)), c)))

## S3 method for class 'zoo'
simpleSmoothTs(x, ..., n = NROW(x))
```

Arguments

`x, y`	data points. Should define a regular, ordered series. A time series object can be passed as the first argument, in which case `y` can be omitted. The `x` argument given to `simpleSmoothTs` is allowed to be a multivariate time series, i.e. to have multiple columns.

`...`	further arguments passed on to `panel.lines`.
`width`	nominal width of the smoothing kernel in time steps. In the default case, which is a simple moving average, this is the actual width. When `c > 1` the number of time steps used in the kernel increases but the equivalent bandwidth stays the same. If only past values are used (with `sides = 1`) then `width` refers to one side of the symmetric kernel.
`n`	approximate number of time steps desired for the result. If this is less than the length of `x`, the smoothed time series will be aggregated by averaging blocks of (an integer number of) time steps, and this aggregated series will be centered with respect to the original series.
`c`	smoothness of the kernel: `c = 1` is a moving average, `c = 2` is a triangular kernel, `c = 3` and higher approximate smooth Gaussian kernels. `c` is actually the number of times to recursively convolve a simple moving average kernel with itself. The kernel size is adjusted to maintain a constant equivalent bandwidth as `c` increases.
`sides`	if `sides=1` the smoothed series is calculed from past values only (using one half of the symmetric kernel); if `sides=2` it is centred around lag 0.
`circular`	to treat the data as circular (periodic).
`kern`	a `tskernel` object; if given, this over-rides `width` and `c`.

Note

The author is not an expert on time series theory.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`kernel`, `filter`, `xyplot.ts`

Examples

```
## a Gaussian-like filter (contrast with c = 1 or c = 2)
xyplot(sunspot.year) +
layer(panel.tskernel(x, y, width = 20, c = 3, col = 1, lwd = 2))

## example from ?kernel:
## long and short moving averages, backwards in time
xyplot(EuStockMarkets[,1]) +
  layer(panel.tskernel(x, y, width = 100, col = 1, sides = 1)) +
  layer(panel.tskernel(x, y, width = 20, col = 2, sides = 1))

## per group, with a triangular filter
xyplot(EuStockMarkets, superpose = TRUE) +
  glayer(panel.tskernel(..., width = 100, c = 2),
         theme = simpleTheme(lwd = 2))
```

```
## plot the actual kernels used; note adjustment of width
width = 100
kdat <- lapply(1:4, function(c) {
    k <- kernel("daniell", rep(floor(0.5*width / sqrt(c)), c))
    ## demonstrate that the effective bandwidth stays the same:
    message("c = ", c, ": effective bandwidth = ", bandwidth.kernel(k))
    ## represent the kernel as a time series, for plotting
    ts(k[-k$m:k$m], start = -k$m)
})
names(kdat) <- paste("c =", 1:4)
xyplot(do.call(ts.union, kdat), type = "h",
    scales = list(y = list(relation = "same")))
```

`panel.voronoi` *Panel functions for level-coded irregular points*

Description

These panel functions for `levelplot` can represent irregular (x, y) points with a color covariate. `panel.levelplot.points` simply draws color-coded points. `panel.voronoi` uses the **deldir** package to calculate the spatial extension of a set of points in 2 dimensions. This is known variously as a Voronoi mosaic, a Dirichlet tesselation, or Thiessen polygons.

Usage

```
panel.voronoi(x, y, z, subscripts = TRUE, at = pretty(z),
   points = TRUE, border = "transparent",
   na.rm = FALSE, win.expand = 0.07, use.tripack = FALSE,
   ...,
   col.regions = regions$col, alpha.regions = regions$alpha)

panel.levelplot.points(x, y, z, subscripts = TRUE, at = pretty(z),
   shrink, labels, label.style, contour, region,
   pch = 21, col.symbol = "#00000044",
   ...,
   col.regions = regions$col, fill = NULL)
```

Arguments

`x, y, z`	an irregular set of points at locations (x, y) with value z.
`subscripts`	integer vector indicating what subset of x, y and z to draw. Typically passed by `levelplot`.
`at, col.regions, alpha.regions`	color scale definition; see `panel.levelplot`.
`points`	whether to draw the (x, y) points.
`border`	color for polygon borders.

`na.rm`	if `TRUE`, points with missing `z` values will be excluded from the calculation of polygons. If `FALSE`, those polygons are calculated but are not drawn (i.e. are transparent).
`win.expand`	defines the rectangular window bounding the polygons. This is a factor by which to expand the range of the data. Set to 0 to limit drawing at the furthest data point locations. Ignored if `use.tripack = TRUE`.
`use.tripack`	if `TRUE`, use **tripack** package rather than **deldir**. See Details.
`...`	further arguments are passed to `panel.xyplot` if `points = TRUE`.
`pch, col.symbol`	symbol and border color for points. A filled symbol should be used, i.e. in the range 21-25.
`shrink, labels, label.style, contour, region, fill`	ignored.

Details

The **tripack** package implementation is faster than **deldir** but not under a fully free licence. Also, the **deldir** package allows polygons to be clipped to a rectangular window (the `win.expand` argument).

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

`tileplot`, `panel.levelplot`, `deldir`

Examples

```
## a variant of Figure 5.6 from Sarkar (2008)
## http://lmdvr.r-forge.r-project.org/figures/figures.html?chapter=05;figure=05_06

depth.ord <- rev(order(quakes$depth))
quakes$Magnitude <- equal.count(quakes$mag, 4)
quakes.ordered <- quakes[depth.ord, ]

levelplot(depth ~ long + lat | Magnitude, data = quakes.ordered,
          panel = panel.levelplot.points, type = c("p", "g"),
          aspect = "iso", prepanel = prepanel.default.xyplot)

## a levelplot with jittered cells

xyz <- expand.grid(x = 0:9, y = 0:9)
xyz[] <- jitter(as.matrix(xyz))
xyz$z <- with(xyz, sqrt((x - 5)^2 + (y - 5)^2))
levelplot(z ~ x * y, xyz, panel = panel.voronoi, points = FALSE)

## hexagonal cells
```

```
xyz$y <- xyz$y + c(0, 0.5)
levelplot(z ~ x * y, xyz, panel = panel.voronoi, points = FALSE)
```

`panel.xblocks` *Plot contiguous blocks along x axis.*

Description

Plot contiguous blocks along x axis. A typical use would be to highlight events or periods of missing data.

Usage

```
panel.xblocks(x, ...)

## Default S3 method:
panel.xblocks(x, y, ..., col = NULL, border = NA,
              height = unit(1, "npc"),
              block.y = unit(0, "npc"), vjust = 0,
              name = "xblocks", gaps = FALSE,
              last.step = median(diff(tail(x))))

## S3 method for class 'ts'
panel.xblocks(x, y = x, ...)
## S3 method for class 'zoo'
panel.xblocks(x, y = x, ...)
```

Arguments

`x, y`	In the default method, `x` gives the ordinates along the x axis and must be in increasing order. `y` gives the color values to plot as contiguous blocks. If `y` is numeric, data coverage is plotted, by converting it into a logical (`!is.na(y)`). Finally, if `y` is a function, it is applied to `x` (`time(x)` in the time series methods). If `y` has character (or factor) values, these are interpreted as colors – and should therefore be color names or hex codes. Missing values in `y` are not plotted. The default color is taken from the current theme: `trellis.par.get("plot.line")$col`. If `col` is given, this over-rides the block colors. The `ts` and `zoo` methods plot the `y` values against the time index `time(x)`.
`...`	In the default method, further arguments are graphical parameters passed on to `gpar`.
`col`	if `col` is specified, it determines the colors of the blocks defined by `y`. If multiple colors are specified they will be repeated to cover the total number of blocks.
`border`	border color.
`height`	height of blocks, defaulting to the full panel height. Numeric values are interpreted as native units.

block.y	y axis position of the blocks. Numeric values are interpreted as native units.
vjust	vertical justification of the blocks relative to block.y. See grid.rect.
name	a name for the grob (**grid** object).
gaps	Deprecated. Use panel.xblocks(time(z),is.na(z)) instead.
last.step	width (in native units) of the final block. Defaults to the median of the last 5 time steps (assuming steps are regular).

Details

Blocks are drawn forward in "time" from the specified x locations, up until the following value. Contiguous blocks are calculated using rle.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

xyplot.ts, panel.rect, grid.rect

Examples

```
## Example of highlighting peaks in a time series.
set.seed(0)
flow <- ts(filter(rlnorm(200, mean = 1), 0.8, method = "r"))

## using an explicit panel function
xyplot(flow, panel = function(x, y, ...) {
  panel.xblocks(x, y > mean(y), col = "lightgray")
  panel.xyplot(x, y, ...)
})
## using layers; this is the `ts` method because `>` keeps it as ts.
xyplot(flow) +
  layer_(panel.xblocks(flow > mean(flow), col = "lightgray"))

## Example of alternating colors, here showing calendar months
flowdates <- as.Date("2000-01-01") + as.numeric(time(flow))
xyplot(flow ~ flowdates, type = "l") +
  layer_(panel.xblocks(x, months,
         col = c("lightgray", "#e6e6e6"), border = "darkgray"))

## highlight values above and below thresholds.
## blue, gray, red colors:
bgr <- hcl(c(0, 0, 260), c = c(100, 0, 100), l = c(90, 90, 90))
dflow <- cut(flow, c(0,15,30,Inf), labels = bgr)
xyplot(flow) + layer_(panel.xblocks(time(flow), dflow))

## Example of highlighting gaps (NAs) in time series.
## set up example data
z <- ts(cbind(A = 0:5, B = c(6:7, NA, NA, 10:11), C = c(NA, 13:17)))
```

```
## show data coverage only (highlighting gaps)
xyplot(z, panel = panel.xblocks,
       scales = list(y = list(draw = FALSE)))

## draw gaps in darkgray
xyplot(z, type = c("p","s")) +
  layer_(panel.xblocks(x, is.na(y), col = "darkgray"))

## Example of overlaying blocks from a different series.
## Are US presidential approval ratings linked to sunspot activity?
## Set block height, default justification is along the bottom.
xyplot(presidents) + layer(panel.xblocks(sunspot.year > 50, height = 2))
```

`panel.xyarea` *Plot series as filled polygons.*

Description

Plot series as filled polygons connected at given origin level (on y axis).

Usage

```
panel.xyarea(x, ...)

## Default S3 method:
panel.xyarea(x, y, groups = NULL, origin = NULL, horizontal = FALSE,
             col, col.line, border, lty, lwd, alpha, ...,
             fill, panel.groups = panel.xyarea)

## S3 method for class 'ts'
panel.xyarea(x, y = x, ...)
## S3 method for class 'zoo'
panel.xyarea(x, y = x, ...)

panel.qqmath.xyarea(x, y = NULL, f.value = NULL, distribution = qnorm,
                    qtype = 7, groups = NULL, ..., tails.n = 0)
```

Arguments

`x, y`	data vectors.
`groups`	a factor defining groups.
`origin`	level on y axis to connect the start and end of the series. If `NULL`, the polygon is filled to the bottom of the panel. It is flipped if `horizontal = TRUE`.
`horizontal`	if this is set to `TRUE`, then the origin is a level on the x axis, rather than the default which is on the y axis. This is the opposite of what you might expect, but is for consistency with `panel.xyplot`.

col, col.line, border, lty, lwd, alpha
: graphical parameters taken from trellis.par.get("plot.polygon") or trellis.par.get("superpos
(when groups defined). col.line overrides col.

...
: further arguments passed on to panel.polygon. For panel.qqmath.xyarea, passed to panel.xyarea.

fill
: ignored; use col instead.

panel.groups
: used in panel.superpose.

f.value, distribution, qtype, tails.n
: as in panel.qqmath.

Details

none yet.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

panel.xyplot, panel.polygon

Examples

```
xyplot(sunspot.year, panel = panel.xyarea, origin = 0,
  aspect = "xy", cut = list(n = 3, overlap = 0))

## two series superposed: one filled, one as a line.
xyplot(ts.union(data = sunspot.year, lag10 = lag(sunspot.year, 10)),
  aspect = "xy", cut = list(n = 3, overlap = 0),
  superpose = TRUE,
  panel = panel.superpose,
  panel.groups = function(..., group.number) {
    if (group.number == 1)
      panel.xyarea(...) else panel.xyplot(...)
  }, border = NA,
  par.settings = simpleTheme(col = c("grey", "black"), lwd = c(5,2)))

## missing values are handled by splitting the series
tmp <- window(sunspot.year, start = 1900)
tmp[c(1:2, 50:60)] <- NA
xyplot(tmp, panel = panel.xyarea, origin = 0)

set.seed(0)
qqmath(~ data, make.groups(baseline = rnorm(100),
                           other = rnorm(100) * 2 - 0.5),
  groups = which, distribution = qunif,
  panel = panel.qqmath.xyarea, xlim = 0:1,
  auto.key = list(points = FALSE, rectangles = TRUE),
  par.settings = simpleTheme(col = c("blue", "green"),
                             alpha = 0.5))
```

panel.xyimage *Scatter plots using emoji-like images*

Description

Panel function that create scatter plots with emoji-like images for plotting character. Images can be local files or URLs. Only PNG and JPEG images are currently supported.

Usage

```
panel.xyimage(x, y,
              subscripts,
              groups = NULL,
              pch = NULL,
              cex = 1,
              ...,
              grid = FALSE, abline = NULL)
```

Arguments

x, y	Data vectors to be plotted.
subscripts	An integer vector of subscripts giving indices of the x and y values in the original data source. See the corresponding entry in xyplot for details.
groups	A factor defining groups.
pch	A character vector giving path(s) or URL(s) of PNG or JPEG files. If groups is specified, there should be one value for each level, otherwise this should have length 1. Recycled as necessary.
cex	A numeric multiplier for the size of the symbols. As with pch, this can be a vector corresponding to levels of groups.
...	Further arguments are accepted but ignored.
grid	A logical flag, character string, or list specifying whether and how a background grid should be drawn. See panel.xyplot for details.
abline	A numeric vector or list, specifying arguments arguments for panel.abline, which is called with those arguments. See panel.xyplot for details.

Details

The image sources given by pch are downloaded if necessary and read in every time the panel function is called. If the same images are to be used in multiple panels, it may be more efficient to download them once and provide the file paths rather than provide URLs.

Author(s)

Deepayan Sarkar

See Also

panel.xyplot

Examples

```
## Not run:
alive <- "https://twemoji.maxcdn.com/72x72/1f60a.png"
dead <- "https://twemoji.maxcdn.com/72x72/1f480.png"

dotplot(Titanic,
        scales = list(x = "free"),
        between = list(x = 1),
        panel = panel.xyimage,
        pch = c(dead, alive), grid = "h",
        main = "Survival on the Titanic",
        xlab = "Number of persons")

## End(Not run)
```

---

postdoc *Reasons for Taking First Postdoctoral Appointment*

Description

Reasons for Taking First Postdoctoral Appointment, by Field of Doctrate, 1997

Usage

```
data(postdoc)
```

Format

The data set is avaliable as a two-way table of counts.

Source

Survey of Doctorate Recipients, 1997

References

Enhancing the Postdoctoral Experience for Scientists and Engineers: A Guide for Postdoctoral Scholars, Advisers, Institutions, Funding Organizations, and Disciplinary Societies

http://books.nap.edu/catalog.php?record_id=9831

Examples

```
data(postdoc)
library(lattice)
barchart(prop.table(postdoc, margin = 1),
         auto.key = TRUE, xlab = "Proportion")
```

`resizePanels` *Resize panels to match data scales*

Description

Modify a `"trellis"` object so that when plotted, the panels have the specified relative width and height.

Usage

```
resizePanels(x, h = 1, w = 1)
```

Arguments

`x`	An object of class `"trellis"`.
`h`	numeric vector specifying panel heights
`w`	numeric vector specifying of panel widths

Details

`resizePanels` modifies a `"trellis"` object so that when plotted, the panels have the specified relative width and height; this is only interesting when `h` or `w` are vectors with unequal entries. `resizePanels` can be called with no arguments, in which case the currently plotted `"trellis"` object (if any) is used for `x`, and a suitable `h` or `w` (based on the current panel layout) is chosen so that sizes are relative to the current panel ranges in the native coordinate system. This is only interesting when `scales="free"`; the resulting object, when plotted again, will have varying panel sizes but the same number of data units per inch in all panels.

Value

An object of class `"trellis"`; essentially the same as `x`, but with certain properties modified.

Author(s)

Deepayan Sarkar

See Also

`Lattice`, `xyplot`

Examples

```
state <- data.frame(state.x77, state.region, state.name)
state$state.name <-
    with(state, reorder(reorder(state.name, Frost),
                        as.numeric(state.region)))
dpfrost <-
    dotplot(state.name ~ Frost | reorder(state.region, Frost),
            data = state, layout = c(1, 4),
            scales = list(y = list(relation = "free")))

## approximate
resizePanels(dpfrost,
             h = with(state, table(reorder(state.region, Frost))))

## exact (including boundary padding)
resizePanels()
```

---

rootogram *Trellis Displays of Tukey's Hanging Rootograms*

Description

Displays hanging rootograms.

Usage

```
rootogram(x, ...)

## S3 method for class 'formula'
rootogram(x, data = parent.frame(),
          ylab = expression(sqrt(P(X == x))),
          prepanel = prepanel.rootogram,
          panel = panel.rootogram,
          ...,
          probability = TRUE)

prepanel.rootogram(x, y = table(x),
                   dfun = NULL,
                   transformation = sqrt,
                   hang = TRUE,
                   probability = TRUE,
                   ...)

panel.rootogram(x, y = table(x),
                dfun = NULL,
                col = plot.line$col,
```

```
lty = plot.line$lty,
lwd = plot.line$lwd,
alpha = plot.line$alpha,
transformation = sqrt,
hang = TRUE,
probability = TRUE,
type = "l", pch = 16,
...)
```

Arguments

x, y	For rootogram, x is the object on which method dispatch is carried out. For the "formula" method, x is a formula describing the form of conditioning plot. The formula can be either of the form ~x or of the form y~x. In the first case, x is assumed to be a vector of raw observations, and an observed frequency distribution is computed from it. In the second case, x is assumed to be unique values and y the corresponding frequencies. In either case, further conditioning variables are allowed. A similar interpretation holds for x and y in prepanel.rootogram and panel.rootogram. Note that the data are assumed to arise from a discrete distribution with some probability mass function. See details below.
data	For the "formula" method, a data frame containing values for any variables in the formula, as well as those in groups and subset if applicable (groups is currently ignored by the default panel function). By default the environment where the function was called from is used.
dfun	a probability mass function, to be evaluated at unique x values
prepanel, panel	panel and prepanel function used to create the display.
ylab	the y-axis label; typically a character string or an expression.
col, lty, lwd, alpha	graphical parameters
transformation	a vectorized function. Relative frequencies (observed) and theoretical probabilities (dfun) are transformed by this function before being plotted.
hang	logical, whether lines representing observed relative freuqncies should "hang" from the curve representing the theoretical probabilities.
probability	A logical flag, controlling whether the y-values are to be standardized to be probabilities by dividing by their sum.
type	A character vector consisting of one or both of "p" and "l". If "p" is included, the evaluated values of dfun will be denoted by points, and if "l" is included, they will be joined by lines.
pch	The plotting character to be used for the "p" type.
...	extra arguments, passed on as appropriate. Standard lattice arguments as well as arguments to panel.rootogram can be supplied directly in the high level rootogram call.

Details

This function implements Tukey's hanging rootograms. As implemented, `rootogram` assumes that the data arise from a discrete distribution (either supplied in raw form, when `y` is unspecified, or in terms of the frequency distribution) with some unknown probability mass function (p.m.f.). The purpose of the plot is to check whether the supplied theoretical p.m.f. `dfun` is a reasonable fit for the data.

It is reasonable to consider rootograms for continuous data by discretizing it (similar to a histogram), but this must be done by the user before calling `rootogram`. An example is given below.

Also consider the `rootogram` function in the `vcd` package, especially if the number of unique values is small.

Value

`rootogram` produces an object of class `"trellis"`. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Deepayan Sarkar <deepayan.sarkar@gmail.com>

References

John W. Tukey (1972) Some graphic and semi-graphic displays. In T. A. Bancroft (Ed) *Statistical Papers in Honor of George W. Snedecor*, pp. 293–316. Available online at http://www.edwardtufte.com/tufte/tukey

See Also

xyplot

Examples

```
library(lattice)

x <- rpois(1000, lambda = 50)

p <- rootogram(~x, dfun = function(x) dpois(x, lambda = 50))
p

lambdav <- c(30, 40, 50, 60, 70)

update(p[rep(1, length(lambdav))],
       aspect = "xy",
       panel = function(x, ...) {
           panel.rootogram(x,
                           dfun = function(x)
                           dpois(x, lambda = lambdav[panel.number()]))
       })
```

```
lambdav <- c(46, 48, 50, 52, 54)

update(p[rep(1, length(lambdav))],
       aspect = "xy",
       prepanel = function(x, ...) {
           tmp <-
               lapply(lambdav,
                      function(lambda) {
                          prepanel.rootogram(x,
                                             dfun = function(x)
                                             dpois(x, lambda = lambda))
                      })
           list(xlim = range(sapply(tmp, "[[", "xlim")),
                ylim = range(sapply(tmp, "[[", "ylim")),
                dx = do.call("c", lapply(tmp, "[[", "dx")),
                dy = do.call("c", lapply(tmp, "[[", "dy")))
       },
       panel = function(x, ...) {
           panel.rootogram(x,
                           dfun = function(x)
                           dpois(x, lambda = lambdav[panel.number()]))
           grid::grid.text(bquote(Poisson(lambda == .(foo)),
                                  where = list(foo = lambdav[panel.number()])),
                           y = 0.15,
                           gp = grid::gpar(cex = 1.5))
       },
       xlab = "",
       sub = "Random sample from Poisson(50)")


## Example using continuous data

xnorm <- rnorm(1000)

## 'discretize' by binning and replacing data by bin midpoints

h <- hist(xnorm, plot = FALSE)

## Option 1: Assume bin probabilities proportional to dnorm()

norm.factor <- sum(dnorm(h$mids, mean(xnorm), sd(xnorm)))

rootogram(counts ~ mids, data = h,
          dfun = function(x) {
              dnorm(x, mean(xnorm), sd(xnorm)) / norm.factor
          })

## Option 2: Compute probabilities explicitly using pnorm()

pdisc <- diff(pnorm(h$breaks, mean = mean(xnorm), sd = sd(xnorm)))
pdisc <- pdisc / sum(pdisc)
```

```
rootogram(counts ~ mids, data = h,
          dfun = function(x) {
              f <- factor(x, levels = h$mids)
              pdisc[f]
          })
```

`scale.components` *Custom lattice axis scales*

Description

Convenience functions for drawing axes with various non-default tick positions and labels.

Usage

```
xscale.components.logpower(lim, ...)
yscale.components.logpower(lim, ...)

xscale.components.fractions(lim, logsc = FALSE, ...)
yscale.components.fractions(lim, logsc = FALSE, ...)

xscale.components.log10ticks(lim, logsc = FALSE, at = NULL, ...)
yscale.components.log10ticks(lim, logsc = FALSE, at = NULL, ...)

xscale.components.log(lim, logsc = FALSE, at = NULL, loc = NULL, ...)
yscale.components.log(lim, logsc = FALSE, at = NULL, loc = NULL, ...)

xscale.components.log10.3(lim, logsc = FALSE, at = NULL, ...)
yscale.components.log10.3(lim, logsc = FALSE, at = NULL, ...)

xscale.components.subticks(lim, ..., n = 5, n2 = n * 5, min.n2 = n + 5)
yscale.components.subticks(lim, ..., n = 5, n2 = n * 5, min.n2 = n + 5)
```

Arguments

`lim`	scale limits.
`...`	passed on to `xscale.components.default` or `yscale.components.default`.
`logsc`	log base, typically specified in the `scales` argument to a high-level lattice plot.
`at`	this is ignored unless it is `NULL`, in which case nothing is drawn.
`loc`	Ignored if `NULL`. Otherwise, vector of values between 1 and 9 that form the template for tick mark locations after being scaled appropriately. For example, `loc = c(1,3)` leads to tick marks at `c(0.1,0.3,1,3,10,30)` and so on.
`n`	desired number of intervals between major axis ticks (passed to `pretty`.
`n2, min.n2`	desired, and minimum, number of intervals between minor axis ticks (passed to `pretty`.

Details

These functions are intended to be passed to the `xscale.components` or `yscale.components` arguments of high-level lattice plots. See `xscale.components.default`.

`xscale.components.logpower` draws tickmarks at the same locations as the default, but labels them more smartly using superscripts for the power (using expressions).

`xscale.components.fractions` labels the tickmarks as fractions.

`xscale.components.log10ticks` puts major tick marks at powers of 10, and minor tickmarks in between.

References

Sarkar, Deepayan (2008) "Lattice: Multivariate Data Visualization with R", Springer. ISBN: 978-0-387-75968-5 http://lmdvr.r-forge.r-project.org/figures/figures.html?chapter=08;figure=08_04

Examples

```
xyplot((1:200)/20 ~ (1:200)/20, type = c("p", "g"),
  scales = list(x = list(log = 2), y = list(log = 10)),
  xscale.components = xscale.components.fractions,
  yscale.components = yscale.components.log10ticks)

xyplot((1:200)/20 ~ (1:200)/20, type = c("p", "g"),
  scales = list(x = list(log = 2), y = list(log = 10)),
  xscale.components = xscale.components.logpower,
  yscale.components = yscale.components.log10.3)

dd <- as.Date("2000-01-01") + 0:365
xyplot(0:365 ~ dd, type = c("p", "g"),
  xscale.components = xscale.components.subticks,
  yscale.components = yscale.components.subticks)
```

SeatacWeather *Daily Rainfall and Temperature at the Seattle-Tacoma Airport*

Description

Daily Rainfall and Temperature at the Seattle-Tacoma Airport between January through March of 2007.

Usage

```
data(SeatacWeather)
```

Format

A data frame with 90 observations on the following 14 variables.

`month` a factor with levels `January`, `February`, and `March`

`day` day of the month

`year` year, all 2007

`max.temp` maximum temperature (Fahrenheit)

`record.max` record maximum temperature

`normal.max` normal maximum temperature

`min.temp` minimum temperature

`record.min` record minimum temperature

`normal.min` normal minimum temperature

`precip` precipitation (inches)

`record.precip` record precipitation

`normal.precip` normal precipitation

`time.max` time of maximum temperature

`time.min` time of minimum temperature

Details

The time of minimum and maximum temperatures should be interpreted as follows: the least two significant digits denote minutes (out of 60) and the next two significant digits denote hout (out of 24).

Source

http://www.atmos.washington.edu/cgi-bin/list_climate.cgi?clisea

---

segplot *Plot segments using the Trellis framework*

Description

This function can be used to systematically draw segments using a formula interface to produce Trellis displays using the lattice package. Segments can be drawn either as lines or bars, and can be color coded by the value of a covariate, with a suitable legend.

Usage

```
segplot(x, data, ...)

## S3 method for class 'formula'
segplot(x, data,
        level = NULL, centers = NULL,
        prepanel = prepanel.segplot,
        panel = panel.segplot,
        xlab = NULL, ylab = NULL,
        horizontal = TRUE,
        ...,
        at, cuts = 30, colorkey = !is.null(level))
```

Arguments

x	Argument on which argument dispatch is carried out. For the "formula" method, a formula of the form y ~ x1 + x2 (with further conditioning variables appended if necessary). The terms in the formula must all be vectors of the same length. Each element causes a line segment or rectangle to be drawn, with the vertical location determined by y and horizontal endpoints determined by x1 and x2.
data	An optional data frame, list or environment where variables in the formula, as well as level, will be evaluated.
level	An optional covariate that determines color coding of the segments
centers	optional vector of 'centers' of the segments. If specified, points will be plotted at these y-locations.
prepanel	function determining range of the data rectangle from data to be used in a panel.
panel	function to render the graphic given the data. This is the function that actually implements the display.
xlab, ylab	Labels for the axes. By default both are missing.
horizontal	logical, whether the segments are to be drawn horizontally (the default) or vertically. This essentially swaps the role of the x- and y-axes in each panel.
...	further arguments. Arguments to levelplot as well as to the default panel function panel.segplot can be supplied directly to segplot.
colorkey	logical indicating whether a legend showing association of segment colors to values of level should be shown, or a list to control details of such a color key. See details below.
at, cuts	at specifies the values of level where the color code changes. If at is missing, it defaults to cuts equispaced locations spanning the range of levels

Details

The levelplot function from the lattice package is used to internally to implement this function. In particular, the colorkey mechanism is used as it is, and documentation for levelplot should be consulted to learn how to fine tune it.

Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Note

Currently only horizontal segments are supported. Vertical segments can be obtained by modifying the prepanel and panel functions suitably.

Author(s)

Deepayan Sarkar <deepayan.sarkar@r-project.org>

See Also

Lattice, panel.segplot, levelplot, xyplot

Examples

```
segplot(factor(1:10) ~ rnorm(10) + rnorm(10), level = runif(10))

data(USCancerRates)

segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male,
        data = subset(USCancerRates, state == "Washington"))

segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male,
        data = subset(USCancerRates, state == "Washington"),
        draw.bands = FALSE, centers = rate.male)

segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male,
        data = subset(USCancerRates, state == "Washington"),
        level = rate.female, col.regions = terrain.colors)

segplot(reorder(factor(county), rate.male) ~ LCL95.male + UCL95.male,
        data = subset(USCancerRates, state == "Washington"),
        draw.bands = FALSE, centers = rate.male,
        segments.fun = panel.arrows, ends = "both",
        angle = 90, length = 1, unit = "mm")
```

---

theEconomist.theme *Generate plots with style like The Economist.*

Description

Uses colors, drawing styles, axis settings, etc, to produce plots inspired by those in The Economist magazine. (http://www.economist.com/).

Usage

```
theEconomist.theme(win.fontfamily = NULL,
    with.bg = FALSE, box = "black", ...)

theEconomist.opts()

asTheEconomist(x, ...,
    type = "l",
    ylab = expression(NULL),
    xlab = expression(NULL),
    par.settings =
      theEconomist.theme(with.bg = with.bg, box = "transparent"),
    with.bg = FALSE,
    par.strip.text = list(font = 2))
```

Arguments

win.fontfamily	on Windows systems, sets the font by name.
with.bg	if TRUE, uses a light blue background and a few other corresponding changes; otherwise white.
box	color for panel boxes, strip outlines, and axis ticks.
...	further arguments passed to simpleTheme and used to modify the theme.
x	a trellis object, i.e. the result of a high-level plot function in the Lattice framework.
type	plot type, relevant for xyplots, see panel.xyplot.
ylab, xlab	axis labels, blank by default.
par.settings	style settings, defaulting to theEconomist.theme.
par.strip.text	see xyplot.

Details

You can just use par.settings = theEconomist.theme(), which gives you some colors and styles, but it does not do the grid lines or axis settings.

Author(s)

Felix Andrews <felix@nfrac.org>

References

http://www.economist.com/displayStory.cfm?story_id=15065782

http://www.economist.com/displayStory.cfm?story_id=14941181

See Also

custom.theme

Examples

```
xyplot(window(sunspot.year, start = 1900),
  main = "Sunspot cycles", sub = "Number per year",
  par.settings = theEconomist.theme(box = "transparent"),
  lattice.options = theEconomist.opts())

asTheEconomist(xyplot(window(sunspot.year, start = 1900),
  main = "Sunspot cycles", sub = "Number per year"))

trellis.last.object() +
  layer_(panel.xblocks(x, x >= 1980, col = "#6CCFF6", alpha = .5)) +
  layer(panel.text(1988, 180, "Forecast", font = 3, pos = 2))

## set as defaults -- remember to set back when finished.
opar <- trellis.par.get()
trellis.par.set(theEconomist.theme(box = "transparent"))
oopt <- lattice.options(theEconomist.opts())

barchart(Titanic[,,,"No"], main = "Titanic deaths", layout = 1:2,
    sub = "by sex and class", auto.key = list(columns = 2),
    scales = list(y = list(alternating = 2)))

asTheEconomist(
  dotplot(VADeaths, main = "Death Rates in Virginia (1940)",
    auto.key = list(corner = c(.9,0.1))),
  type = "b", with.bg = TRUE)

dotplot(VADeaths, auto.key = TRUE, type = "b",
  par.settings = theEconomist.theme(with.bg = TRUE))

asTheEconomist(
  densityplot(~ height, groups = voice.part, data = singer,
    subset = grep("1", voice.part), plot.points = FALSE)) +
glayer(d <- density(x), i <- which.max(d$y),
  ltext(d$x[i], d$y[i], paste("Group", group.number), pos = 3))

## reset
trellis.par.set(opar)
lattice.options(oopt)
```

---

tileplot *Plot a spatial mosaic from irregular 2D points*

Description

Represents an irregular set of (x, y) points with a color covariate. Polygons are drawn enclosing the area closest to each point. This is known variously as a Voronoi mosaic, a Dirichlet tesselation, or Thiessen polygons.

Usage

```
tileplot(x, data = NULL, aspect = "iso",
         prepanel = "prepanel.default.xyplot",
         panel = "panel.voronoi", ...)
```

Arguments

x, data	formula and data as in levelplot, except that it expects irregularly spaced points rather than a regular grid.
aspect	aspect ratio: "iso" is recommended as it reproduces the distances used in the triangulation calculations.
panel, prepanel	see xyplot.
...	further arguments to the panel function, which defaults to panel.voronoi.

Details

See panel.voronoi for further options and details.

Author(s)

Felix Andrews <felix@nfrac.org>

See Also

panel.voronoi, levelplot

Examples

```
xyz <- data.frame(x = rnorm(100), y = rnorm(100), z = rnorm(100))
tileplot(z ~ x * y, xyz)

## tripack is faster but non-free
## Not run:
tileplot(z ~ x * y, xyz, use.tripack = TRUE)

## End(Not run)

## showing rectangular window boundary
tileplot(z ~ x * y, xyz, xlim = c(-2, 4), ylim = c(-2, 4))

## insert some missing values
xyz$z[1:10] <- NA
## the default na.rm = FALSE shows missing polygons
tileplot(z ~ x * y, xyz, border = "black",
  col.regions = grey.colors(100),
  pch = ifelse(is.na(xyz$z), 4, 21),
  panel = function(...) {
    panel.fill("hotpink")
    panel.voronoi(...)
```

```
  })
## use na.rm = TRUE to ignore points with missing values
update(trellis.last.object(), na.rm = TRUE)

## a quick and dirty approximation to US state boundaries
tmp <- state.center
tmp$Income <- state.x77[,"Income"]
tileplot(Income ~ x * y, tmp, border = "black",
  panel = function(x, y, ...) {
    panel.voronoi(x, y, ..., points = FALSE)
    panel.text(x, y, state.abb, cex = 0.6)
  })
```

USAge *US national population estimates*

Description

US national population estimates by age and sex from 1900 to 1979. The data is available both as a (3-dimensional) table and a data frame. The second form omits the 75+ age group to keep age numeric.

Usage

```
data(USAge.table)
data(USAge.df)
```

Format

`USAge.table` is a 3-dimensional array with dimensions

No	Name	Levels
1	Age	0, 1, 2, . . . , 74, 75+
2	Sex	Male, Female
3	Year	1900, 1901, . . . , 1979

Cells contain raw counts of estimated population.

`USAge.df` is a data frame with 12000 observations on the following 4 variables.

`Age` a numeric vector, giving age in years

`Sex` a factor with levels `Male` `Female`

`Year` a numeric vector, giving year

`Population` a numeric vector, giving population in millions

Details

The data for 1900-1929 are rounded to thousands. The data for 1900-1939 exclude the Armed Forces overseas and the population residing in Alaska and Hawaii. The data for 1940-1949 represent the resident population plus Armed Forces overseas, but exclude the population residing in Alaska and Hawaii. The data for 1950-1979 represent the resident population plus Armed Forces overseas, and also include the population residing in Alaska and Hawaii.

Source

U.S. Census Bureau website: http://www.census.gov/

The data were available as individual files for each year, with varying levels for the margins. The preprocessing steps used to reduce the data to the form given here are described in the scripts directory.

Examples

```
data(USAge.df)
head(USAge.df)

## Figure 10.7 from Sarkar (2008)
xyplot(Population ~ Age | factor(Year), USAge.df,
       groups = Sex, type = c("l", "g"),
       auto.key = list(points = FALSE, lines = TRUE, columns = 2),
       aspect = "xy", ylab = "Population (millions)",
       subset = Year %in% seq(1905, 1975, by = 10))
```

USCancerRates *Rate of Death Due to Cancer in US Counties*

Description

This data set records the annual rates of death (1999-2003) due to cancer by sex in US counties.

Usage

```
data(USCancerRates)
```

Format

A data frame with 3041 observations on the following 8 variables.

`rate.male` a numeric vector, giving rate of death per 100,000 due to cancer among males

`LCL95.male` a 95% lower confidence limit for `rate.male`

`UCL95.male` a 95% upper confidence limit for `rate.male`

`rate.female` a numeric vector, giving rate of death per 100,000 due to cancer among females

`LCL95.female` a 95% lower confidence limit for `rate.female`

`UCL95.female` a 95% upper confidence limit for `rate.female`

`state` a factor with levels giving name of US state

`county` a character vector giving county names, in a format similar to that used for county map boundaries in the `maps` package.

Details

See the scripts directory for details of data preprocessing steps.

From the website: Death data provided by the National Vital Statistics System public use data file. Death rates calculated by the National Cancer Institute using SEER*Stat. Death rates are age-adjusted to the 2000 US standard population [http://www.seer.cancer.gov/stdpopulations/stdpop.19ages.html]. Population counts for denominators are based on Census populations as modified by NCI.

Source

http://statecancerprofiles.cancer.gov/

Examples

```
data(USCancerRates)
```

`useOuterStrips` *Put Strips on the Boundary of a Lattice Display*

Description

Try to update a `"trellis"` object so that strips are only shown on the top and left boundaries when printed, instead of in every panel as is usual. This is only meaningful when there are exactly two conditioning variables.

Usage

```
useOuterStrips(x,
               strip = strip.default,
               strip.left = strip.custom(horizontal = FALSE),
               strip.lines = 1,
               strip.left.lines = strip.lines)
```

Arguments

x	An object of class "trellis".
strip, strip.left	A function, character string or logical that would be appropriate strip and strip.left arguments respectively in a high level lattice function call (see xyplot)
strip.lines, strip.left.lines	height of strips in number of lines; helpful for multi-line text or mathematical annotation in strips.

Details

useOuterStrips modifies a "trellis" object with length(dim(x)) == 2 so that when plotted, strips are only shown on the top and left boundaries of the panel layout, rather than on top of every panel, as is the usual behaviour.

If the original "trellis" object x includes non-default strip and strip.left arguments, they will be ignored. To provide customized strip behaviour, specify the custom strip functions directly as arguments to useOuterStrips.

Value

An object of class "trellis"; essentially the same as x, but with certain properties modified.

Author(s)

Deepayan Sarkar

See Also

Lattice, xyplot

Examples

```
library(lattice)

mtcars$HP <- equal.count(mtcars$hp)

useOuterStrips(xyplot(mpg ~ disp | HP + factor(cyl), mtcars))

useOuterStrips(xyplot(mpg ~ disp | factor(cyl) + HP, mtcars),
               strip.left = FALSE,
               strip = strip.custom(style = 4))
```

`xyplot.stl` *Display stl fits with Lattice*

Description

Display `stl` decomposition (seasonal, trend and irregular components using loess) with Lattice, like the base graphics function `plot.stl`.

Usage

```
## S3 method for class 'stl'
xyplot(x, data = NULL,
       outer = TRUE,
       layout = c(1, 4),
       strip = FALSE,
       strip.left = TRUE,
       as.table = TRUE,
       ylab = "",
       between = list(y = 0.5),
       panel =
       function(..., type) {
           if (packet.number() == 4) type <- "h"
           panel.xyplot(..., type = type)
       },
       ...)
```

Arguments

`x`	an `stl` object.
`data`	ignored.
`outer, layout, strip, strip.left`	passed to `xyplot.ts`.
`as.table, ylab, between, panel, ...`	passed to `xyplot.ts`.

Details

Unless `strip.left` is passed in explicitly, a custom strip will be drawn, where shaded bars are comparable across panels (representing the same data range).

Value

An object of class `"trellis"`. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Author(s)

Deepayan Sarkar <deepayan.sarkar@r-project.org>

See Also

stl, xyplot.ts, xyplot

Examples

```
## example from ?stl
xyplot(stl(log(co2), s.window=21),
       main = "STL decomposition of CO2 data")

## alternative display
data(biocAccess)
xyplot(stl(ts(biocAccess$counts[1:(24 * 30)], frequency = 24), "periodic"),
       strip.left = "strip.default")
resizePanels()

## two different spans
xyplot(stl(nottem, s.window = 4)) +
as.layer(xyplot(stl(nottem, s.window = "periodic")), style = 2)

## components superposed
xyplot(stl(nottem, s.window = 4), superpose=TRUE,
  screens = list(data = "trend", trend = "trend", "residuals"),
  strip.left = TRUE, layout = c(1,2))
```

Index

∗Topic **aplot**
as.layer, 3
c.trellis, 6
doubleYScale, 14
layer, 28
panel.key, 42
panel.lmlineq, 43
panel.scaleArrow, 48
∗Topic **datasets**
ancestry, 2
biocAccess, 5
EastAuClimate, 16
gvhd10, 23
postdoc, 62
SeatacWeather, 69
USAge, 76
USCancerRates, 77
∗Topic **dplot**
combineLimits, 9
custom.theme, 11
dendrogramGrob, 12
ecdfplot, 19
ggplot2like.theme, 21
panel.2dsmoother, 37
panel.3dmisc, 38
panel.ellipse, 41
panel.qqmath.tails, 46
panel.quantile, 47
panel.segplot, 50
panel.smoother, 51
panel.tskernel, 53
panel.xblocks, 57
panel.xyarea, 59
panel.xyimage, 61
resizePanels, 63
rootogram, 64
scale.components, 68
theEconomist.theme, 72
useOuterStrips, 78
∗Topic **hplot**
horizonplot, 24
mapplot, 32
marginal.plot, 35
panel.voronoi, 55
segplot, 70
tileplot, 74
xyplot.stl, 80
∗Topic **ts**
horizonplot, 24
xyplot.stl, 80
+.trellis *(*layer*)*, 28
[.layer *(*layer*)*, 28

ancestry, 2
as.layer, 3, *15*, *30*
asTheEconomist *(*theEconomist.theme*)*, 72
axis.default, *21*
axis.grid *(*ggplot2like.theme*)*, 21

biocAccess, 5

c.trellis, 6
cloud, *40*
cm.colors, *33*
colorRampPalette, *11*, *21*
combineLimits, 9
cov.trob, *41*
custom.theme, 11, *22*, *73*

deldir, *56*
dendrogram, *12*
dendrogramGrob, 12
doubleYScale, *4*, 14
draw.key, *43*
drawLayer *(*layer*)*, 28

EastAuClimate, 16
ecdfplot, 19

filter, *54*

flattenPanel (layer), 28

ggplot2like (ggplot2like.theme), 21
ggplot2like.theme, 21
glayer (layer), 28
glayer_ (layer), 28
gpar, *13*, *57*
grid.rect, *58*
grid.text, *49*
gvhd10, 23

hcl, *21*
heatmap, *13*
horizonplot, 24

kernel, *54*

Lattice, *10*, *26*, *34*, *63*, *72*, *79*
lattice.options, *21*
layer, *4*, 28
layer_ (layer), 28
levelplot, *13*, *25*, *37*, *51*, *55*, *71, 72*, *75*
lm, *44, 45*
loess, *37, 38*

mapplot, *3*, 32
marginal.plot, *7*, 35
mergedTrellisLegendGrob (c.trellis), 6

panel.2dsmoother, 37
panel.3dbars (panel.3dmisc), 38
panel.3dmisc, 38
panel.3dpolygon (panel.3dmisc), 38
panel.3dtext (panel.3dmisc), 38
panel.abline, *43–45*, *49*, *61*
panel.ablineq (panel.lmlineq), 43
panel.arrows, *49*, *51*
panel.axis, *4*
panel.cloud, *40*
panel.densityplot, *36*
panel.dotplot, *36*
panel.ecdfplot (ecdfplot), 19
panel.ellipse, 41
panel.horizonplot, *25*
panel.horizonplot (horizonplot), 24
panel.key, 42
panel.levelplot, *37*, *55, 56*
panel.levelplot.points (panel.voronoi), 55
panel.lines, *54*
panel.lmline, *43*
panel.lmlineq, 43
panel.loess, *52*
panel.mapplot (mapplot), 32
panel.polygon, *60*
panel.qqmath, *20*, *46*, *60*
panel.qqmath.tails, 46
panel.qqmath.xyarea (panel.xyarea), 59
panel.quantile, 47, *52*
panel.rect, *58*
panel.rootogram (rootogram), 64
panel.scaleArrow, 48
panel.segments, *51*
panel.segplot, 50, *71, 72*
panel.smoother, *38*, *48*, 51
panel.superpose, *29*, *60*
panel.text, *44, 45*, *49*
panel.tskernel, 53
panel.voronoi, 55, *75*
panel.xblocks, 57
panel.xyarea, *26*, 59
panel.xyimage, 61
panel.xyplot, *20*, *41*, *46*, *56*, *60–62*, *73*
plot.stl, *80*
plotmath, *44, 45*
postdoc, 62
predict.rq, *47*
prepanel.ecdfplot (ecdfplot), 19
prepanel.horizonplot (horizonplot), 24
prepanel.mapplot (mapplot), 32
prepanel.rootogram (rootogram), 64
prepanel.segplot (panel.segplot), 50
pretty, *68*
print, *26*, *72*, *80*
print.layer (layer), 28
print.trellis, *6, 7*

qqmath, *20*, *46*

resizePanels, 63
rle, *58*
rootogram, 64
rq, *47, 48*

scale.components, *22*, 68
SeatacWeather, 69
segplot, *51*, 70
simpleKey, *15*, *42, 43*
simpleSmoothTs (panel.tskernel), 53

simpleTheme, *11*, *21*, *73*
stat_quantile, *47*, *48*
stat_smooth, *51*, *52*
stl, *80*, *81*

theEconomist.axis *(*theEconomist.theme*)*, 72
theEconomist.opts *(*theEconomist.theme*)*, 72
theEconomist.theme, 72
tileplot, *56*, 74
trellis.currentLayout, *29*
trellis.device, *11*
trellis.object, *7*
trellis.par.get, *11*
trellis.par.set, *21*, *29*

unit, *13*, *49*
update, *26*, *72*, *80*
update.trellis, *7*, *30*
USAge, 76
USCancerRates, 77
useOuterStrips, 78

xscale.components.default, *68*, *69*
xscale.components.fractions *(*scale.components*)*, 68
xscale.components.log *(*scale.components*)*, 68
xscale.components.log10.3 *(*scale.components*)*, 68
xscale.components.log10ticks *(*scale.components*)*, 68
xscale.components.logpower *(*scale.components*)*, 68
xscale.components.subticks *(*scale.components*)*, 68
xyplot, *6*, *10*, *25*, *33*, *36*, *43*, *61*, *63*, *66*, *72*, *73*, *75*, *79*, *81*
xyplot.list *(*c.trellis*)*, 6
xyplot.stl, 80
xyplot.ts, *25*, *26*, *54*, *58*, *80*, *81*

yscale.components.fractions *(*scale.components*)*, 68
yscale.components.log *(*scale.components*)*, 68
yscale.components.log10.3 *(*scale.components*)*, 68
yscale.components.log10ticks *(*scale.components*)*, 68
yscale.components.logpower *(*scale.components*)*, 68
yscale.components.subticks *(*scale.components*)*, 68