

Package ‘latticeExtra’

October 27, 2009

Version 0.6-4

Date 2009/10/27

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 Extra graphical utilities based on lattice

Depends R (>= 2.5.0), RColorBrewer, lattice

Imports lattice (>= 0.16-3), grid

Suggests maps, mapproj, deldir, tripack

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

LazyLoad yes

LazyData no

License GPL (>= 2)

Repository CRAN

Date/Publication 2009-10-27 07:23:33

R topics documented:

ancestry	2
as.layer	3
biocAccess	5
c.trellis	5
custom.theme	8
dendrogramGrob	9
doubleYScale	11
EastAuClimate	13

ecdfplot	16
gplot	17
gplotArgs.data.frame	18
gvhd10	20
layer	21
mapplot	23
marginal.plot	25
panel.3dmisc	27
panel.lmlineq	29
panel.qqmath.tails	32
panel.segplot	33
panel.voronoi	34
panel.xblocks	36
panel.xyarea	37
postdoc	39
rootogram	39
SeatacWeather	42
segplot	43
tileplot	45
USAge	47
USCancerRates	48
useOuterStrips	49

Index	51
--------------	-----------

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:

http://factfinder.census.gov/jsp/saff/SAFFInfo.jsp?_pageId=sp4_decennial_sf3

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 with different scales

Description

Allows overlaying of lattice plots with different scales. The overlaid plots include custom axes and may be drawn in a different style.

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, ...)
```

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 <code>scales\$draw == FALSE</code> .
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.
...	passed to layer : typically the <code>style</code> 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 settings 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

```
## all same scales:
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(mdeaths ~ 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)
plot(stl(ts(biocAccess$counts[1:(24 * 30)], frequency = 24), "periodic"))
```

c.trellis *Merge trellis objects*

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, recursive = FALSE)

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

Arguments

<code>...</code>	two or more trellis objects. If these are named arguments, the names will be used in the corresponding panel strips.
<code>x.same</code>	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.
<code>y.same</code>	as above, for y scales. Note that <code>xyplot.list</code> defaults to same y scales. Set to NA to leave them alone.
<code>layout</code>	value for layout of the new plot; see <code>xyplot</code> .
<code>recursive</code>	for consistency with the generic method, ignored.
<code>x</code>	a list; the function FUN, which defaults to <code>xyplot</code> , will be called on each element of x, and the resulting plots combined into one.
<code>FUN, data</code>	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, legends, 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))
## suppress scales on the first 3 panels
update(qua, scales = list(at = list(NULL, NULL, NULL, NA),

```

```

y = list(draw = FALSE))

## 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)
statemap$names <- gsub(".*", "", statemap$names)
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,
  prepanel = function(...)
    list(xlim = c(0, max(prepanel.default.densityplot(...)$xlim)),
  scales = list(relation = "free", x = list(axes = "i")),
  cex = 0.5, ref = TRUE)
## 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))

```

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

```

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

Value

A list that can be supplied to [trellis.par.get](#) or as the `theme` argument to [trellis.device](#).

Author(s)

Deepayan Sarkar

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 <code>order.dendrogram(x)</code> , 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"))))

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",
                              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, add.axis = TRUE,
             add.ylab2 = FALSE, style1 = 1, style2 = 2,
             text = NULL, auto.key = if (!is.null(text))
               list(text, points = points, lines = lines, ...),
             points = FALSE, lines = TRUE, ...)

```

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 | if TRUE, set plot style (with <code>trellis.par.set</code>) to different style settings for each series. The settings are taken mainly from <code>trellis.par.get("superpose.line")</code> . These will also be applied to the y-axes and ylab, if relevant. |
| 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"`.

`style1, style2` if `use.style` is TRUE, these give the ‘group number’ for `obj1` and `obj2` respectively. The style is taken from these indices into the values of `trellis.par.get("superpose")`. Therefore these should be integers between 1 and 6; a value of 0 can be given to leave the default settings.

`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.

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"))

## 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/averages/> 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, 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",
        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

<code>x</code>	For <code>ecdfplot</code> , <code>x</code> is the object on which method dispatch is carried out. For the "formula" method, <code>x</code> is a formula describing the form of conditioning plot, and has to be of the form <code>~x</code> , where <code>x</code> is assumed to be a numeric vector. Further conditioning variables are allowed as usual. A similar interpretation holds for <code>x</code> in the "numeric" method as well as <code>prepanel.ecdfplot</code> and <code>panel.ecdfplot</code> .
<code>data</code>	For the "formula" method, a data frame containing values for any variables in the formula, as well as those in <code>groups</code> and <code>subset</code> if applicable.
<code>prepanel, panel</code>	panel and prepanel function used to create the display.
<code>xlab, ylab</code>	axis labels; typically a character string or an expression.
<code>groups</code>	a grouping variable of the same length as <code>x</code> . If specified, ECDF plots are computed for each subset defined by unique values of <code>groups</code> and the resulting functions superposed within each panel.
<code>f.value, qtype</code>	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 <code>panel.ecdfplot</code> can be supplied directly in the high level <code>ecdfplot</code> 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)
```

gplot

Grouped Display

Description

Generic plotting function to produce grouped display using lattice

Usage

```
gplot(x, ...)
gplotArgs(x, ...)
gplotArgs(x) <- value
## Default S3 method:
gplot(x, ...)
## Default S3 method:
gplotArgs(x, ...)
```

Arguments

x	an arbitrary object (typically one with a suitable method)
value	list
...	extra arguments, passed on as appropriate

Details

later.

Value

`gplot` should produce 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.

`gplotArgs` should return a list.

Author(s)

Deepayan Sarkar <deepayan@stat.wisc.edu>

See Also

[Lattice](#)

`gplotArgs.data.frame`

Grouped Display Arguments from a Data Frame

Description

Extract arguments suitable for `gplot` from a `data.frame` with appropriate meta-data

Usage

```
## S3 method for class 'data.frame':
gplotArgs(x, display.formula,
          outer = FALSE, inner = FALSE, ...)
## S3 replacement method for class 'data.frame':
gplotArgs(x) <- value
```

Arguments

<code>x</code>	an object of class “data.frame”
<code>display.formula</code>	a Trellis display formula
<code>outer</code>	logical or one-sided formula
<code>inner</code>	logical or one-sided formula
<code>...</code>	extra arguments, passed on as appropriate
<code>value</code>	a list containing named components, to be stored as special attributes of <code>x</code> , including but not limited to

formula: a model formula with a response, covariate and grouping variable, i.e. of the form $y \sim x \mid g$, where x could be 1

order.groups: logical, defaults to TRUE, whether levels of g will be reordered for the plot

FUN: a function of x used to reorder levels of g , applied to subsets of x determined by g

outer: one-sided formula, defaults to NULL

inner: one-sided formula, defaults to NULL

labels: list of character strings (or expressions?) with named components, specifying variable names used in plot labels. The component names must be variables in the data frame. Not all variables need be present (if not, the variable names will be used for labels).

units: list, similar to `labels`, but specifying the units of measurement (if present, used in constructing labels)

These are modeled on the `groupedData` function in package `nlme`, and are stored as meta-data in the data frame `x`. The components listed above are special in the sense that they are used by the corresponding `gplotArgs` method to construct a suitable list as described in `gplotArgs.default`. Any other components in `value` can be used to override the values thus constructed, including `display.formula`.

Details

The `gplotArgs` method for data frames constructs a suitable list as described in `gplotArgs.default`. The `display.formula` argument can be used to specify the Trellis formula used (usually, it will be constructed from the meta-data).

For the role of `outer` and more details, consult `nlme` documentation for now.

Value

`gplotArgs` should return a list.

Author(s)

Deepayan Sarkar <deepayan@stat.wisc.edu>

See Also[gplot](#)**Examples**

```
gplotArgs(iris) <-  
  list(formula = Sepal.Length ~ 1 | Species,  
        order.groups = TRUE,  
        labels = list(Sepal.Length = "Sepal\nLength",  
                      Sepal.Width = "Sepal\nWidth"))  
gplot(iris)  
gplot(iris, display.formula = Sepal.Length ~ Sepal.Width | Species)
```

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://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**

layer

*Conveniently augment trellis objects***Description**

A mechanism to add new layers to a trellis object, optionally using a new data source. This works by adding arbitrary expressions to the panel function. It can simplify programmatic augmentation of trellis objects.

Usage

```
layer(..., data = NULL, under = FALSE, style = NULL)

## S3 method for class 'trellis':
x + lay

flattenPanel(x)
```

Arguments

<code>...</code>	expressions as they would appear in a panel function. These can refer to the panel function arguments (typically <code>x</code> , <code>y</code> and <code>subscripts</code>), or alternatively <code>...</code> , which represents all panel function arguments. In addition, <code>data</code> is used as the evaluation frame.
<code>data</code>	an optional data source (<code>data.frame</code> , <code>environment</code> , etc: see eval). The value <code>TRUE</code> can be given, in which case the <code>data</code> argument from the lattice call is used.
<code>under</code>	whether the layer should be drawn below the existing panel.
<code>style</code>	style index of the layer, used only to set lattice graphical parameters (same effect as in grouped displays).
<code>x</code>	a trellis object.
<code>lay</code>	a layer object.

Details

The `layer` mechanism is an alternative method for editing the 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 programmatic augmentation of trellis objects.

A typical example would be 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 passed in via the `data` object: `layer(panel.refline(h = myVal), data = list(myVal = myVal))`. Alternatively, `eval(bquote(layer(panel.refline(h = .(myVal))))`.

`layer()` is an experimental function. It probably should not be used in package code, since it does not fit well with the trellis model, and the implementation is fairly inefficient. A better option for package code is usually a custom panel function.

When the `data` argument is `TRUE`, an attempt is made to extract the object passed as a `data` argument in the original lattice plot call.

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 to be edited manually.

Value

a `layer` object is defined as a list of expression objects, each of which may have a `data` attribute. 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)

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

## underlay a flat colour
foo <- foo + layer(panel.fill(grey(.9)), 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

## see an outline of the complete panel function
flattenPanel(foo)

## layers with superposed styles
zoip <- 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)
update(zoip, key = simpleKey(c("span = 0.5", "span = 1.0")),
  title = "loess smooth", lines = TRUE, points = FALSE)

## 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 = TRUE)

## 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(gridlon, gridlat, dist,
  contour = TRUE, subscr = TRUE), data = newdat, under = TRUE)

```

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(brewer.pal(n = 11, name = "Spectral")),
  colorkey = TRUE,
  ...)

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

```

Arguments

<code>x, y</code>	For <code>mapplot</code> , an object on which method dispatch is carried out. For the formula method, a formula of the form $y \sim x$, with additional conditioning variables as desired. The extended form of conditioning using $y \sim x1 + x2$ etc. is also allowed. The formula might be interpreted as in a dot plot, except that <code>y</code> is taken to be the names of geographical units in <code>map</code> . Suitable subsets (packets) of <code>x</code> and <code>y</code> are passed to the <code>prepanel</code> and <code>panel</code> functions.
<code>data</code>	A data source where names in the formula are evaluated
<code>map</code>	An object of class "map" (package <code>maps</code>), containing boundary information. The names of the geographical units must match the <code>y</code> variable in the formula.
<code>outer</code>	logical; how variables separated by <code>+</code> in the formula are interpreted. It is not advisable to change the default.
<code>prepanel, panel</code>	the <code>prepanel</code> and <code>panel</code> functions
<code>aspect</code>	aspect ratio
<code>breaks, cuts, colramp</code>	controls conversion of numeric <code>x</code> values to a false color. <code>colramp</code> may be a vector of colors or a function that produces colors (such as <code>cm.colors</code>)
<code>legend, colorkey</code>	controls legends; usually just a color key giving the association between numeric values of <code>x</code> and color.
<code>lwd</code>	line width
<code>...</code>	Further arguments passed on to the underlying engine. See <code>xyplot</code> 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)

data(USCancerRates)

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

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")

mapplot(county ~ log10(population), ancestry, map = county.map)

## 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 = 1,
              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

<code>x</code>	a data frame or table, or a formula of which the first term is a data frame or table. Otherwise coerced with <code>as.data.frame</code> .
<code>data</code>	an optional data source in which groups and subset may be evaluated.
<code>groups</code>	term, to be evaluated in <code>data</code> , that is used as a grouping variable.
<code>reorder</code>	whether to reorder factor variables by frequency.
<code>subset</code>	data subset expression, evaluated in <code>data</code> .
<code>plot.points</code> , <code>ref</code> , <code>cut</code>	passed to <code>panel.densityplot</code> .
<code>origin</code> , <code>type</code>	passed to <code>panel.dotplot</code> .
<code>xlab</code> , <code>ylab</code> , <code>as.table</code> , <code>subscripts</code>	see xyplot .
<code>default.scales</code> , <code>layout</code> , <code>lattice.options</code>	see xyplot .
<code>...</code>	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.3dmisc

Miscellaneous panel utilities for three dimensional Trellis Displays

Description

Miscellaneous 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

`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 sides of the bars

`...` 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(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.lmlineq`

Draw a line and its equation.

Description

This is an extension of the panel functions [panel.abline](#) and [panel.lmline](#) to also draw the equation of a line.

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`, `adj`, `fontfamily`
passed on to [panel.text](#).
For `pos`: 1 = below, 2 = left, 3 = above, 4 = right, and the `offset` (in character widths) is applied.
For `adj`: `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`.
- `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`, `sep`, `sep.end`
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.

The R^2 value is separated from the equation by `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(),
       key = simpleKey(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).

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](#)**Examples**

```
xx <- rt(10000, df = 10)
c(panel.qqmath = qqmath(~xx),
  approx = qqmath(~xx, panel = panel.qqmath.tails))
```

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.regions = regions$col,
              band.height = 0.6,
              horizontal = TRUE,
              ...,
              centers = NULL,
              pch = 16)
```

Arguments

<code>x, y, z</code>	Vectors corresponding to <code>x1, x2</code> and <code>y</code> respectively in the <code>segplot</code> formula. The names are different for compatibility with <code>panel.levelplot</code> . These are all the original vectors in <code>data</code> , not subsetted for particular panels.
<code>level</code>	optional vector controlling color of segments
<code>centers</code>	optional vector of ‘centers’ of the segments. If specified, points will be plotted at these <code>y</code> -locations.
<code>pch</code>	plotting character used for <code>centers</code> .
<code>subscripts</code>	integer subscript to be used as an indexing vector for <code>x, y, z</code> and <code>level</code> , 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. 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.regions vector of colors as in [levelplot](#)

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

... other arguments, passed to panel.rect or panel.segments as appropriate.

Value

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

Author(s)

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

See Also

[segplot](#)

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 tessellation, 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` colour 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.

Usage

```

panel.xblocks(x, ...)

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

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

```

Arguments

<code>x</code> , <code>y</code>	In the default method, <code>x</code> gives the ordinates along the x axis and must be in increasing order. <code>y</code> gives the colour values to plot as contiguous blocks. These may be character (or factor) values (colour names or hex codes), numbers (indexing <code>palette()</code>), or logicals (where <code>TRUE</code> is <code>palette(1)</code> and <code>FALSE</code> is transparent). Missing values in <code>y</code> are not plotted. See also <code>col</code> , below, which over-rides values of <code>y</code> . In the <code>ts</code> method, plot values against their <code>time()</code> , unless <code>y</code> is specified, when it acts just like the default method.
<code>...</code>	In the default method, further arguments are graphical parameters passed on to gpar .
<code>height</code>	height of blocks, defaulting to the full panel height. Numeric values are interpreted as native units.

<code>block.y</code>	y axis position of the blocks. Numeric values are interpreted as native units.
<code>vjust</code>	vertical justification of the blocks relative to <code>block.y</code> . See grid.rect .
<code>col</code>	if <code>col</code> is specified, then all values of <code>y</code> that are not NA or FALSE are drawn in this colour.
<code>border</code>	border colour.
<code>name</code>	a name for the grob (grid object).
<code>last.step</code>	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 by [rle](#).

Author(s)

Felix Andrews (felix@nfrac.org)

See Also

[panel.rect](#), [grid.rect](#)

Examples

```
## Are US presidential approval ratings linked to sunspot activity?

## 'ts' method; set block height, default justification is at bottom.
xyplot(presidents) + layer(panel.xblocks(sunspot.year > 50, height = 3))

## blocks in a light shading colour, full panel height.
xyplot(presidents) +
  layer(panel.xblocks(sunspot.year > 50, col = "#e6e6e6"), under=TRUE)

## multiple colour values given in the 'y' argument.
sscols <- cut(sunspot.year, c(50,150,Inf), labels=c("yellow","orange"))
xyplot(presidents, lwd = 2) +
  layer(panel.xblocks(time(sunspot.year), y = sscols, alpha = 0.5))
```

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

Description

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

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 available 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")
```

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,
          ...)

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

```

Arguments

<code>x, y</code>	For <code>rootogram</code> , <code>x</code> is the object on which method dispatch is carried out. For the "formula" method, <code>x</code> is a formula describing the form of conditioning plot. The formula can be either of the form $\sim x$ or of the form $y \sim x$. In the first case, <code>x</code> is assumed to be a vector of raw observations, and an observed frequency distribution is computed from it. In the second case, <code>x</code> is assumed to be unique values and <code>y</code> the corresponding frequencies. In either case, further conditioning variables are allowed. A similar interpretation holds for <code>x</code> and <code>y</code> in <code>prepanel.rootogram</code> and <code>panel.rootogram</code> .
<code>data</code>	For the "formula" method, a data frame containing values for any variables in the formula, as well as those in <code>groups</code> and <code>subset</code> if applicable (<code>groups</code> is currently ignored by the default panel function). By default the environment where the function was called from is used.
<code>dfun</code>	a probability mass function, to be evaluated at unique <code>x</code> values
<code>prepanel, panel</code>	panel and prepanel function used to create the display.
<code>ylab</code>	the y-axis label; typically a character string or an expression.


```

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)")

```

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 hour (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

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

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](#), [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)
```

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 tessellation, or Thiessen polygons.

Usage

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

Arguments

<code>x</code> , <code>data</code>	formula and data as in levelplot , except that it expects irregularly spaced points rather than a regular grid.
<code>aspect</code>	aspect ratio: "iso" is recommended as it reproduces the distances used in the triangulation calculations.
<code>panel</code> , <code>prepanel</code>	see xyplot .
<code>...</code>	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

<http://www.census.gov/popest/archives/pre-1980/PE-11.html> U.S. Census Bureau, Population Division. Internet Release date: October 1, 2004

The data were available as individual files for 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.table)
```

USCancerRates	<i>Rate of Death Due to Cancer in US Counties</i>
---------------	---

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

Tried 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)

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

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.
h	numeric vector specifying panel heights
w	numeric vector specifying of panel widths

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.

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

```
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))

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()
```

Index

*Topic **aplot**

as.layer, 3
c.trellis, 5
doubleYScale, 11
layer, 21
panel.lmlineq, 29

*Topic **datasets**

ancestry, 2
biocAccess, 4
EastAuClimate, 13
gvhd10, 20
postdoc, 39
SeatacWeather, 42
USAge, 47
USCancerRates, 48

*Topic **dplot**

custom.theme, 8
dendrogramGrob, 9
ecdfplot, 15
gplot, 17
gplotArgs.data.frame, 18
panel.3dmisc, 27
panel.qqmath.tails, 32
panel.segplot, 33
panel.xblocks, 36
panel.xyarea, 37
rootogram, 39
useOuterStrips, 49

*Topic **hplot**

mapplot, 23
marginal.plot, 25
panel.voronoi, 34
segplot, 43
tileplot, 45

+ .trellis (layer), 21

ancestry, 2
as.layer, 3, 12, 22

biocAccess, 4

c.trellis, 5

cloud, 29

cm.colors, 24

colorRampPalette, 8

custom.theme, 8

deldir, 35

dendrogram, 9

dendrogramGrob, 9

doubleYScale, 4, 11

EastAuClimate, 13

ecdfplot, 15

eval, 21

flattenPanel (layer), 21

gpar, 9, 36

gplot, 17, 19

gplotArgs (gplot), 17

gplotArgs.data.frame, 18

gplotArgs.default, 19

gplotArgs<- (gplot), 17

gplotArgs<- .data.frame
(gplotArgs.data.frame), 18

grid.rect, 37

gvhd10, 20

heatmap, 10

Lattice, 18, 24, 45, 50

layer, 3, 4, 21

levelplot, 10, 34, 35, 44–46

lm, 30, 31

mapplot, 2, 23

marginal.plot, 6, 25

panel.3dbars (panel.3dmisc), 27

panel.3dmisc, 27

panel.3dpolygon (panel.3dmisc), 27

panel.3dtext (*panel.3dmisc*), 27
 panel.abline, 29–31
 panel.ablineq (*panel.lmlineq*), 29
 panel.axis, 4
 panel.cloud, 29
 panel.densityplot, 26, 27
 panel.dotplot, 26, 27
 panel.ecdfplot (*ecdfplot*), 15
 panel.levelplot, 35
 panel.levelplot.points
 (*panel.voronoi*), 34
 panel.lmline, 29
 panel.lmlineq, 29
 panel.mapplot (*mapplot*), 23
 panel.polygon, 38
 panel.qqmath, 16, 32, 33, 38
 panel.qqmath.tails, 32
 panel.qqmath.xyarea
 (*panel.xyarea*), 37
 panel.rect, 37
 panel.rootogram (*rootogram*), 39
 panel.segplot, 33, 44
 panel.superpose, 38
 panel.text, 30, 31
 panel.voronoi, 34, 45, 46
 panel.xblocks, 36
 panel.xyarea, 37
 panel.xyplot, 16, 32, 35, 38
 plotmath, 30, 31
 postdoc, 39
 prepanel.ecdfplot (*ecdfplot*), 15
 prepanel.mapplot (*mapplot*), 23
 prepanel.rootogram (*rootogram*), 39
 prepanel.segplot (*panel.segplot*),
 33
 print, 44
 print.layer (*layer*), 21
 print.trellis, 6

 qqmath, 17, 32

 resizePanels (*useOuterStrips*), 49
 rle, 37
 rootogram, 39

 SeatacWeather, 42
 segplot, 34, 43
 simpleKey, 11

 tileplot, 35, 45

 trellis.device, 8
 trellis.object, 6
 trellis.par.get, 8

 unit, 9
 update, 44
 update.trellis, 6, 22
 USAge, 47
 USCancerRates, 48
 useOuterStrips, 49

 xyplot, 5, 24, 26, 41, 45, 49, 50
 xyplot.list (*c.trellis*), 5